

Bimanual Robotic Manipulation through Imitation with Deep Generative Models and Expressive Representations

Bimanuale Robotikmanipulation durch Imitation mit tiefen generativen Modellen und ausdrucksstarken Repräsentationen

Master thesis by Nick Striebel

Date of submission: March 27, 2025

1. Review: João Carvalho, Ph.D.
2. Review: Niklas Funk
3. Review: Michael Drolet
4. Review: Prof. Jan Peters, Ph.D.
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



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

Hiermit erkläre ich, Nick Striebel, dass ich die vorliegende Masterarbeit gemäß § 22 Abs. 7 APB 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.

Mannheim, den 27. März 2025



N. Striebel

Abstract

Bimanual robots are essential for solving many real-world tasks typically performed by humans, as numerous manipulation tasks require coordinated actions from two arms. Recently, Imitation Learning has gained renewed attention in robotics due to powerful deep generative machine learning models. These models effectively handle large datasets and address multimodality - two inherent characteristics of human demonstrations, which often contain significant variability and noise.

This thesis presents a structured and extensible framework for defining and generating diverse bimanual manipulation tasks using key point definitions in task space. Leveraging this framework, Behavioral Cloning policies are learned from demonstrated data. Specifically, state-of-the-art deep generative methods, Diffusion and Flow Matching, are utilized. Moreover, Flow Matching is extended to the novel pose-based method ActionFlow [1], introducing explicit object-attention mechanisms and invariance regarding spatial transformations. Extending ActionFlow to a bimanual setting, an aspect previously unexplored, represents a significant portion of this work.

Simulation experiments across environments of varying complexity reveal that ActionFlow achieves superior performance, especially in low-data regimes, tasks with high complexity, long horizons, and special accuracy requirements. This advantage is attributed to ActionFlow's structured pose-based representation. Additionally, empirical evidence explicitly highlights the practical benefits of ActionFlow's invariance to global transformations in bimanual tasks.

Overall, this thesis advances deep generative methods for robot imitation learning, providing novel insights and robust methodologies particularly suited to bimanual robotic setups.

Zusammenfassung

Bimanuelle Roboter sind entscheidend für das Lösen vieler realer Aufgaben, die typischerweise von Menschen ausgeführt werden, da zahlreiche Manipulationsaufgaben koordinierte Aktionen beider Arme erfordern. In den letzten Jahren hat das Imitationslernen in der Robotik aufgrund leistungsstarker tiefer generativer Modelle neue Aufmerksamkeit erfahren. Diese Modelle ermöglichen den effektiven Umgang mit großen Datensätzen und adressieren die Multimodalität menschlicher Demonstrationen, die oft durch starke Variabilität und Rauschen geprägt sind.

Diese Arbeit stellt ein strukturiertes und erweiterbares Framework vor, um vielfältige bimanuelle Manipulationsaufgaben basierend auf Schlüsselpunkt-Definitionen im Aufgabenraum zu generieren. Aufbauend auf diesem Framework werden Strategien mittels Verhaltenklonens (Behavioral Cloning) aus den gesammelten Demonstrationsdaten erlernt. Im Besonderen werden dabei moderne generative Verfahren wie Diffusion und Flow Matching eingesetzt. Zudem wird Flow Matching zur posenbasierten Methode ActionFlow [1] erweitert, die explizite Aufmerksamkeitsmechanismen zwischen Objekten enthält und gegenüber räumlichen Transformationen invariant ist. Die Erweiterung von ActionFlow auf bimanuelle Szenarien - ein bisher nicht behandelte Aspekt - stellt einen bedeutenden Teil dieser Arbeit dar.

Simulationen mit Aufgaben unterschiedlicher Komplexität zeigen, dass ActionFlow eine höhere Performanz erzielt, insbesondere bei geringen Datenmengen, Aufgaben mit hoher Komplexität, langen Zeithorizonten und besonderen Anforderungen an die Genauigkeit. Dieser Vorteil wird insbesondere auf die strukturierte, posenbasierte Repräsentation von ActionFlow zurückgeführt. Darüber hinaus wird der praktische Nutzen der Invarianz von ActionFlow gegenüber globalen Transformationen für bimanuelle Aufgaben empirisch verdeutlicht.

Insgesamt leistet diese Arbeit einen Beitrag zur Weiterentwicklung tiefer generativer Modelle für robotisches Imitationslernen, indem sie neue Erkenntnisse und robuste Methoden insbesondere für bimanuelle Robotersysteme bereitstellt.



Acknowledgments

I want to express my sincere gratitude to my supervisor, João Carvalho, for his dedicated guidance throughout this thesis and continuous support during my previous master's projects. Additionally, I am very grateful to my supervisors, Niklas Funk and Michael Drolet, who significantly enriched this work with their valuable input, ideas, and feedback.

My deep appreciation also goes to Prof. Jan Peters for making this thesis possible, providing continuous access to the Institute for Intelligent Autonomous Systems (IAS) research laboratory throughout my entire master's studies, and giving me access to state-of-the-art robot and computer hardware.

Finally, I want to thank my family and friends for their constant support, encouragement, and assistance.



Software

The code developed during this thesis is hosted in a private GitHub repository. For access and further inquiries, please get in touch with the author at nickstriebel@gmx.de.

Contents

Figures, Tables and Algorithms	vii
Abbreviations	ix
1. Introduction	1
1.1. Contribution and Thesis Structure	2
2. Foundations	3
2.1. Robot Control	3
2.2. Reinforcement Learning	4
2.3. Imitation Learning	6
3. Related Work	8
3.1. Algorithms	8
3.2. Environments & Datasets	10
4. Environment	12
4.1. Robot Control	12
4.2. Action Space	14
4.3. Observation Space	15
5. Bimanual Tasks	16
5.1. Existing Bimanual Tasks	16
5.2. New Bimanual Tasks	17
5.3. Randomization and Task Variations	19
6. Data Collection	20
6.1. Waypoint-Based Expert	20
6.2. Data Storage and Processing	21

7. Learning Framework	23
7.1. Algorithm Overview	23
7.2. Learning Pipeline	24
8. Algorithms	25
8.1. Common Policy Structure	25
8.2. Diffusion Policy	27
8.3. Flow Matching	33
8.4. ActionFlow	35
8.5. Pose-based Flow Matching without IPA	46
9. Experiments	47
9.1. Sanity Check	48
9.2. Sample Efficiency	49
9.3. Invariance of ActionFlow	50
10. Discussion and Outlook	52
References	55
A. Tasks in Detail	I
B. Hyperparameters	VII
C. Experimental Results in Detail	X

Figures, Tables and Algorithms

List of Figures

1.1. Bimanual Tasks	2
5.1. Bimanual tasks of ROBOSUITE	17
5.2. New Bimanual Tasks	18
8.1. Multi-Step Inference Cycle	26
8.2. Forward Diffusion Process.	27
8.3. Diffusion Model	30
8.4. Multi-Head Attention	31
8.5. Probability Flow in 1D	33
8.6. Linear vs. Exponential Schedule in Flow Matching	35
8.7. Flow in $\mathbb{R}^3 \times \text{SO}(3)$	37
8.8. ActionFlow Model	42
8.9. Invariant Point Attention	45
8.10. Pose-based Attention Block without IPA	46
9.1. Success Rates of the Sanity Check Experiments	48
9.2. Success Rates for Varying Numbers of Demonstrations	50
9.3. Success Rates for the Invariance Experiment	51
A.1. Peg-In-Hole Task	I
A.2. Lift Task	II
A.3. Handover Task	III
A.4. Transport Task	III
A.5. Place-Ball Task	IV
A.6. Pick-Place Task	V

A.7. Quad-Insert task	VI
A.8. Hinged-Bin Task	VI
B.1. Transformer Blocks in Comparison	VIII
B.2. Cosine Learning Rate Schedule	IX

List of Tables

5.1. Task Categorization	19
B.1. Datasets and Training Overview	IX
C.1. Results of the Sanity Check Experiment	X
C.2. Results of the Sample Efficiency Experiment	XI
C.3. Results of the Invariance Experiment	XII

List of Algorithms

6.1. Waypoint Expert Rollout	22
8.1. Diffusion Model Training	29
8.2. Diffusion Model Inference	29
8.3. Flow Matching Training	35
8.4. Flow Matching Inference	35
8.5. Training step of a Pose Based Flow Model	40
8.6. Inference step of a Pose-Based Flow Model	40

Abbreviations

Notation	Description
BC	Behavior Cloning
BIL	Bimanual Imitation Learning
CNN	Convolutional Neural Network
GAIL	Generative Adversarial Imitation Learning
i.i.d.	independently and identically distributed
IL	Imitation Learning
IPA	Invariant Point Attention
IRL	Inverse Reinforcement Learning
MDP	Markov Decision Process
MHA	Multi-Head Attention
MLP	Multilayer perceptron
OSC	Operational Space Control

PD	proportional-derivative
PID	proportional–integral–derivative
RL	Reinforcement Learning
SI	International System of Units
SOTA	state-of-the-art

1. Introduction

Robotics has dramatically evolved over recent decades, transforming from rudimentary automated machines into sophisticated, adaptive systems capable of tackling increasingly complex and nuanced tasks once thought exclusively manageable by humans. Among these advancements, bimanual robotics - systems equipped with two manipulators working cooperatively - stands out as promising. They enable the execution of tasks unattainable by single-arm robots. Applications range from precise assembly and medical surgery to complex manufacturing and household chores, highlighting its substantial potential impact on industry, healthcare, and everyday life.

However, despite these impressive capabilities, the manual programming of detailed robotic actions remains prohibitively complex, cumbersome, and often impractical. Human demonstrations inherently capture nuanced strategies and expert insights, offering a rich source of learning data. Consequently, Imitation Learning (IL), which allows robots to acquire complex behaviors directly from observing human or expert demonstrations, has emerged. By leveraging existing expertise, IL significantly simplifies the learning process, removing the need for explicitly defining detailed action sequences or intricate reward structures, thus accelerating development and deployment cycles.

While traditional IL approaches, such as Behavior Cloning (BC), demonstrate considerable potential, they often falter in scenarios that deviate from training data, limiting their practical application. Researchers have approached advanced learning paradigms, including deep generative models, to address these limitations. These models promise better performance and significantly enhanced generalization and adaptability.

This thesis explores deep generative models specifically applied to the Bimanual Imitation Learning (BIL) field. It evaluates cutting-edge algorithms, tests their limitations, and introduces novel modifications designed to substantially boost their robustness, flexibility, and applicability. Through comprehensive experimentation within a structured, modular, and easily extensible learning framework, this work opens pathways toward more versatile, precise, and human-like robotic manipulation.



Figure 1.1.: A selection of bimanual tasks in the developed framework.

1.1. Contribution and Thesis Structure

Chapter 2 covers the foundations of this work and introduces the reader to the relevant notations.

Based on the current state of research and the available sources (Chapter 3), this work is motivated by the need for a structured and flexible approach to BIL and is a twofold contribution:

Development of a Modular (Bimanual) Imitation Learning Framework. The first part of this work (Chapters 4 to 7) focuses on designing a modular framework for BIL. The framework is intended to facilitate the seamless implementation of new environments, simplify data collection, and support the evaluation of various imitation learning algorithms. Researchers can efficiently experiment with different setups and methodologies, as the framework prioritizes modularity and flexibility.

Application of the Framework. In the second part (Chapters 8 and 9), the developed framework is utilized and validated by demonstrating its capabilities with existing methods. Furthermore, it is the foundation for extending ActionFlow [1] to the domain of bimanual - and potentially n -arm - manipulation. ActionFlow is tested against its proposed properties and current state-of-the-art (SOTA) approaches.

This structured approach provides a robust infrastructure for experimentation in imitation learning and contributes to advancing bimanual manipulation research.

Chapter 10 concludes the work with a final summary and an outlook.

2. Foundations

This chapter explores the theoretical fundamentals for subsequent discussions. The reader should note that this chapter only covers the basics, introducing the topic and relevant notation. Deeper insights are given as needed in the following chapters. One important aspect is robot control, a crucial element in robotics that requires accuracy and precision. The key focus is Imitation Learning (IL), which allows robots to learn from demonstrations. Although IL can function independently of Reinforcement Learning (RL), understanding particular RL concepts is beneficial. This chapter provides a brief overview of some of the relevant keys.

2.1. Robot Control

Effective robot control is essential for performing precise tasks. Achieving the necessary accuracy at a low level of control is challenging due to various physical factors. For instance, gravitation must be compensated, and unknown friction within the joints and motors, such as temperature drifts, can complicate achieving smooth and precise control. SOTA controllers must also provide safety properties, like compliance in contact-rich environments.

Generally, two primary approaches to robot control can be distinguished: Control in the joint space and control in the task space. Joint space control involves directly managing the individual angles or positions of each joint in the robot. While this method offers direct control over the robot's internal configuration, it can be challenging to interpret and manage. Alternatively, Operational Space Control (OSC) focuses more intuitively on controlling the robot's end effector (e.g., a robotic arm's gripper or tool) in the task space. The key advantage of this approach is its easy-to-understand and interpretable movement specification. A linear movement in space is specified via a linear trajectory rather than a complex composition of individual joint trajectories. In addition, the control input is

decoupled from the robot’s architecture, allowing it to be applied across different robotic platforms.

2.1.1. Operational Space Control

With OSC [2], [3], it is possible to execute forces in the coordinate system of the actual task, e.g., the world frame. Position and orientation of the robot end effector $\mathbf{x} \in \mathbb{R}^n$ results from the forward kinematics $\mathbf{x} = f_{\text{KINEMATICS}}(\mathbf{q})$ of the robot with its joint coordinates \mathbf{q} . Given the joint velocities $\dot{\mathbf{q}}$, accelerations $\ddot{\mathbf{q}}$ and the Jacobian $\mathbf{J}(\mathbf{q}) = df_{\text{KINEMATICS}}(\mathbf{q})/d\mathbf{q}$ the velocity and acceleration in task space can be derived as

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}}, \quad \ddot{\mathbf{x}} = \dot{\mathbf{J}}(\mathbf{q}) \dot{\mathbf{q}} + \mathbf{J}(\mathbf{q}) \ddot{\mathbf{q}}.$$

OSC allows to translate an arbitrary control signal \mathbf{u} in the task space to the corresponding joint forces $\boldsymbol{\tau}$ [3]:

$$\boldsymbol{\tau} = \mathbf{J}^T(\mathbf{q}) \mathbf{u}. \tag{2.1}$$

In robotics, \mathbf{u} is typically the output of a PID- or proportional-derivative (PD)-controller [4], whereas the first provides higher accuracy, while the latter implements a compliant behavior. Further compliance and control can be achieved by attaching an admittance controller [5].

2.2. Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm where an agent interacts with an environment to achieve a goal by learning from the consequences of its actions. Unlike supervised learning, where a model is trained on a fixed dataset of labeled examples, RL relies on feedback from a reward function guiding the agent toward optimal behavior. RL involves three key components: the agent, the environment, and the reward signal. The agent takes action in the environment, and the environment responds by transitioning to a new state and providing feedback through a reward. Over time, the agent builds a policy that maps states to actions, maximizing its long-term reward.

2.2.1. Task Formulation

Many algorithms model the environment or task as a Markov Decision Process (MDP) [6]. A MDP consists of a state space $\mathbf{s} \in \mathcal{S}$, an action space $\mathbf{a} \in \mathcal{A}$, a transition probability distribution

$$p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1] , \quad \mathbf{s}_{t+1} \sim p(\cdot | \mathbf{s}_t, \mathbf{a}_t) ,$$

an initial state distribution

$$\mu_0 : \mathcal{S} \rightarrow [0, 1] , \quad \mathbf{s}_0 \sim \mu_0 ,$$

and a reward function

$$r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R} . \tag{2.2}$$

Thereby, the index t donates the current time step. The Markov property [6] assumes that the current state \mathbf{s}_t contains all relevant information about the environment and its past:

$$p(\mathbf{s}_{t+1} | \mathbf{s}_{0:t}, \mathbf{a}_{0:t}) = p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) .$$

While this assumption might not match reality perfectly, it has proven to work in many cases.

With this setup, the goal of RL is to find an optimal agent, in the form of a policy π , which outputs an action \mathbf{a} based on the current observation $\mathbf{o} \in \mathcal{O}$. The observation space \mathcal{O} depends on the used sensors and the current state: $\mathcal{S} \xrightarrow{\text{SENSORS}} \mathcal{O}$. Generally, the policy π can be described as a conditional distribution

$$\pi : \mathcal{O} \times \mathcal{A} \rightarrow [0, 1] , \quad \mathbf{a} \sim \pi(\cdot | \mathbf{o}) .$$

Finding the optimal policy is equivalent to maximizing the cumulated reward J_π :

$$\pi^* = \arg \max_{\pi} J_\pi , \quad J_\pi = \mathbb{E}_{\mu_0, p, \pi} \left[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \right] . \tag{2.3}$$

Here, the discounting factor $\gamma \in [0, 1]$ secures finite reward sums.

2.2.2. Difficulties in Reinforcement Learning

Equation (2.3) shows the importance of the reward function r : Its shape implies the ideal behavior. Learning the balance between exploration (trying new actions) and exploitation

(choosing actions known to yield high rewards) is a critical aspect: If the agent acts too greedily at the start, it may not solve the task, but too much exploration results in an unstable learning process. Overcoming these difficulties can be challenging: Depending on the complexity of the task, defining a reward that induces the desired behavior can be arbitrarily complex, and fine-tuning the exploration-exploitation trade-off can be very time-consuming. Furthermore, optimal expert behaviors might have taken years to develop. Cloning such a policy is much easier than learning it from scratch. All this motivates the approach of IL.

2.3. Imitation Learning

IL tries to mimic some expert behavior. It does not need an explicit reward function. Learning from expert demonstrations gives the agent guidance, drastically increasing the learning speed. This comes at the cost of lousy generalization for unseen states: The agent does not know how to return to the experts' trajectory and requires new data in these scenarios. This section provides a collection of various IL approaches. However, it only captures the very basics. The different approaches are classified into the classes *Behavior Cloning* and *Inverse Reinforcement Learning*. This work focuses on the first class. The fundamental working principle is explained in the following section.

2.3.1. Behavior Cloning

BC poses the earliest [7] and purest form of IL. Referring to the MDP setup of RL, the expert demonstrations are given as a set of observation-action traces

$$\mathbf{o}_0 \rightarrow \mathbf{a}_0 \rightarrow \mathbf{o}_1 \rightarrow \mathbf{a}_1 \rightarrow \mathbf{o}_2 \rightarrow \mathbf{a}_2 \rightarrow \dots ,$$

and are called expert trajectories τ_E . BC splits up these trajectories into observation-action pairs $(\mathbf{o}_i, \mathbf{a}_i)$ and treats these pairs to be independently and identically distributed (i.i.d.). With this setup, the BC objective is to learn a policy π_θ that matches the log likelihood of the expert π_E :

$$\hat{\theta} = \arg \max_{\theta} \mathbb{E}_{(\mathbf{o}, \mathbf{a}) \sim \pi_E} [\log \pi_\theta(\mathbf{a} | \mathbf{o})] .$$

Using the Kullback–Leibler divergence [8]

$$\text{KL}(p(x) || q(x)) = \int p(x) \log \frac{p(x)}{q(x)} dx ,$$

this objective can also be formulated via

$$\hat{\theta} = \arg \min_{\theta} \text{KL}(\pi_E(\mathbf{a}|\mathbf{o}) || \pi_{\theta}(\mathbf{a}|\mathbf{o})) . \quad (2.4)$$

In austere environments, BC can work quite well but suffers from the covariate shift and the i.i.d. assumption: Supervised learning assumes that the observation-action pairs $(\mathbf{o}_i, \mathbf{a}_i)$ are distributed i.i.d.. Still, in a MDP, an action in a given state (observation) induces the next state, which breaks this assumption. Errors made in different states can add up until a state out of the experts' distribution is reached. The policy has never been trained in this state (more precisely, on the observation induced by this state). Thus, the behavior is undefined and can lead to failure.

Many SOTA methods [1], [9]–[11] are based on datasets of observation-action pairs. This positions these approaches in the BC rubric. The methods presented in Chapter 8 improve over vanilla BC by using SOTA generative models and by working with action and observation chunks.

3. Related Work

IL is an approach that tries to mimic the behavior of an agent [12]–[14], which proves its capabilities in various robotic tasks [15], [16]. BIL covers the research field of performing human-like tasks with two arms and has acquired a growing interest in recent years [17]–[21]. Early work in this field took a control-based or planning-based approach [22]–[25], while today’s approaches are learning-based and use RL [26]–[28] or Movement Primitives [16], [21], [29]. The latter poses an unique form of IL but comes with a higher inductive bias, as it assumes the learned behavior can be represented through a dynamical system, limiting its flexibility in tasks that require more complex, unstructured, or multimodal behaviors. This work focuses on pure neural network-based approaches trained via IL. An overview of traditional and SOTA IL approaches is given in this section. Existing tasks and benchmarks targeting BIL are presented.

3.1. Algorithms

Nowadays, many algorithms and approaches targeting IL exist. The most basic approach for IL is vanilla BC [7]. It is well-known and simple. BC is a supervised learning approach that targets learning observation-action pairs $\pi(\mathbf{a}|\mathbf{s})$ from expert demonstrations, by assuming that these pairs are i.i.d. The simplicity of this approach comes at the cost of the covariate shift. In out-of-training distribution states, the learned policy is undefined and can lead to failure. These unseen states can be reached easily as the i.i.d. assumption of the action pairs violates the Markov property of MDPs. Nevertheless, BC can be successful if enough expert demonstrations are available.

Dagger [30] extends BC by asking the expert for guidance in unseen states. This reduces the covariate shift and leads to better generalization, but it is only applicable if an expert is available during training.

DART [31] modifies BC without needing an expert during training time. Noise is added

to the experts' policy during data collection, forcing the supervisor to demonstrate how to recover from errors. However, this approach may be impractical in high-precision scenarios, as injecting noise into the demonstrations could result in imminent failure.

Addressing some of the limitations of feed-forward BC, Implicit BC remodels the policy from $\mathbf{a} \sim \pi(\cdot|\mathbf{s})$ to $\mathbf{a} = \arg \min_{\mathbf{a}} E(\mathbf{s}, \mathbf{a})$. E is a learned energy function encoding the expert behavior. On top of handling discontinuities and multi-valued functions, Implicit BC can show better extrapolation in unseen states.

ALOHA [10] showcases its power by using an Action Chunking Transformer and extending BC by learning to predict a sequence of actions as a response to a history of observations. Implementing a Multi-Head Attention (MHA) Transformer [32], ALOHA showed impressive results in real-world scenarios, even in an extended mobile setup [33].

ActionFlow [1] displays another novel approach that works with sequential observations and actions. It generates action sequences by combining Flow Matching [34] with Invariant Point Attention (IPA) [35] to achieve spatial invariance. This method was only tested on single-arm tasks.

Also, using a modern network structure, the Diffusion Policy [9] uses a conditional Diffusion model [36] to learn the gradient of an action-distribution score function, similar to the energy learned in the Implicit BC approach. The main advantage of this method is its improved stability. APEX [19] is another approach that uses Diffusion models but also focuses on collision prevention by adding information about obstacles to the model, making it less general.

Another group of algorithms [11], [18], [37] makes use of the Transformer architecture [32] in combination with RGB-D observations. These methods learn a single model for several tasks by adding a language description. [18] proposes the PerAct² model which explicitly extends the work of [11] to a bimanual setup. [38] is another work that approaches bimanual IL using image data and point clouds. A disadvantage of these methods is the need for large amounts of data, e.g., [18] requires five RGB-D cameras.

While all the previous methods are somehow based on the original BC idea, Generative Adversarial Imitation Learning (GAIL) [39] was the first method approaching the task of IL using Inverse Reinforcement Learning (IRL). IRL approaches try to learn some cost or reward function that explains the experts' behavior. Based on this, the policy is trained using basic RL techniques.

GAIL specifically learns the rewards function as a discriminator component from a General Adversarial Network [40]. The training is completed once the discriminator cannot distinguish between state-action samples from the generator network and the expert. Some adaptations [41]–[43] try to improve this concept further.

Maximum Entropy IRL [44] also learns a reward function that explains the experts'

behavior. The goal is to learn a diverse and multimodal policy by trading this off with maximizing the entropy of the resulting policy. Maximum Causal Entropy IRL [45] additionally considered the entropy of the state distribution over time to take the system dynamics into account. Guided Cost Learning [46] even goes a step further and combines expert demonstrations with active learning to improve the efficiency of learning the reward function.

Adversarial training can suffer from instability between the generator and discriminator. IQ-Learn [47] avoids this by learning a Q -function. The state-action value function encodes the reward and the policy: $\mathbf{a} = \arg \max_{\mathbf{a}'} Q(\mathbf{s}, \mathbf{a}')$. This method proved to work for single-arm manipulators [47].

SQIL [48] also targets the task of imitation using RL and the Q -function. Handing the agent a reward of +1 for matching expert demonstrations and a reward of 0 for all other behaviors, this method aims to match the expert demonstrations over a long horizon. The sparse reward, encoded using a Q function, encourages the agent to stay close to the demonstrated states. Although applying a basic RL approach, this algorithm can be classified as a regularized variant of BC.

3.2. Environments & Datasets

With a large number of developed IL algorithms available, the need for shared environments and benchmarks arises [49]–[55]. However, most works focus on single-arm manipulation: MimicGen [56] allows the production of large-scale datasets with minimal human effort and provides 18 different single-arm tasks. D3IL [57] targets benchmarking with a focus on diverse behaviors in seven single-manipulator tasks. D4ARL [58] contains the Adroit domain [59] and the FrankaKitchen tasks [60], both containing several single-arm tasks. MoMaRT [61] is a collection of five long-horizon robot mobile manipulation tasks in a realistic simulated kitchen. However, the robot only has a single arm with one gripper. RoboTurk [62] enables the collection of large-scale datasets but also only evaluates the framework on three single-arm tasks (*lifting*, *picking*, and *assembly*).

Robosuite [63] is a simulation framework using the MuJoCo [64] physics engine. It provides several environments, four of them implementing bimanual tasks: *handover*, *lift*, *peg-in-hole*, and *transport*.

HumanoidBench [65] focuses on building a simulation benchmark for humanoid robots but also has some static tasks that can be reduced to a bimanual setup.

Providing a large benchmark framework for various manipulation skills in simulations of high realism (based on SAPIEN [66]), ManiSkill [67] contains two bimanual tasks,

namely *move-bucket* and *push-chair*.

ALOHA [10] evaluates its models in five real-world scenarios and the simulated tasks *single-insertion*, *mug-on-plate*, and *double insertion*, all requiring two arms. With Mobile ALOHA [33], the authors add further bimanual tasks with *cook-shrimp*, *wipe-wine*, *use-cabinets*, *rinse-pan* and *push-chairs*. They use a particular setup and build an extensive teleoperation interface.

The authors of [68] implement further bimanual real-world scenarios and benchmarks. It evaluates different cloth manipulation tasks. Bi-KVIL [38] focuses on kitchen tasks like *placing-spoon-on-table*, *pour-water*, and *clean-table*. Bunny-VisionPro [69] provides a teleoperation system that also allows the collection of data for IL; tasks like *pan-cleaning* and *uncovering-and-pouring* are already elaborated. BiDex [20] offers a low-cost and portable bimanual dexterous teleoperation framework to collect data in real-world scenarios that require high dexterity, e.g., *drill-operating* and *wire-winding*. APEX [19] also focuses on real-world tasks and evaluates its method on two bimanual chores: *ball-in-cup* and *vertical-stacking*. BRMData [70] is a dataset for real-world bimanual-mobile robot manipulation in household tasks. Some interesting task implementations are *bottle-handoff*, *wine-wipe*, and *play-with-rubiks-cube*.

Bi-Touch [71] focuses on tactile feedback and uses the tasks *pushing*, *reorienting*, and *gathering* to evaluate their approach. The environments are implemented using PyBullet [72] as the physical backend, making it less potent than works based on MuJoCo [64], for example. The implementation of [73] also uses PyBullet and implements the task of *lift-table-on-box*, presenting a task of a larger scale.

The latest benchmarks for BIL have been established by Drolet et al. [17] and PerAct² [18]. [17] focuses on comparing a cross-section of fundamental and state-of-the-art algorithms. The simulation is based on the work of [29] and uses the MuJoCo [64] physics engine. The experts' data is generated using waypoints. [29] would also enable collecting experts' data with PS Move controllers. [17] implements a single task (*four-peg-insertion-task*) and uses it to benchmark six different algorithms regarding hyperparameter tolerance, noise tolerance, compute efficiency, performance, and training stability. The work uses a minimal observation space without image-like data. [18], on the other hand, only compares vision and RGB-D data-based algorithms. The focus is on training a single model for various tasks using language conditioning. Using CoppeliaSim [74] as its backend, 13 different tasks (e.g., *lift-a-ball*, *handover-an-item*, *straighten-rope*, *put-bottle-in-fridge*) are implemented. These environments extend the RL Bench datasets [53] and use waypoints-based data generation. In contrast to [17], the benchmarking of [18] focuses on analyzing the performance of the algorithms dependent on the coupling (temporal, spatial, and physical) and required symmetrical and synchronized coordination.

4. Environment

The environment setup is the first crucial component of the BIL framework. Since real-world robotic experiments are expensive, time-consuming, and often difficult to reproduce, using simulations as a foundation is the most practical approach. Simulations offer full control over the experimental setup, ensuring repeatability and scalability, making it the best choice for developing and benchmarking learning methods.

When deciding to use simulations, selecting a suitable simulation framework becomes essential. In this work, `ROBOSUITE` [63] is chosen as the primary simulation environment. Built upon the `MuJoCo` [64] physics engine, it provides a solid base. The environment creation is factored from the pure XML-based creation into object-oriented classes that handle the individual components. This gives an intuitive entry point for extensions and adaptations. `ROBOSUITE` provides a rich set of pre-implemented objects and environment components and has built-in support for various robot models. It also offers features like domain randomization, efficient data collection mechanisms, and video recording, and has implemented multiple controller types. Furthermore, it provides fundamental implementations for bimanual manipulation tasks, making it a solid foundation for the framework of this work.

This chapter provides an overview of how `ROBOSUITE` is utilized and introduces the implemented two-arm tasks.

4.1. Robot Control

The control of bimanual robots requires two primary components: (i) the control of the end-effector to manipulate objects and (ii) the control of the gripper to grasp and release objects. The individual components are taken over from `ROBOSUITE`.

4.1.1. Operational Space Control

ROBOSUITE [63] offers several pre-implemented controllers, one of these is the OSC [3]. This controller is chosen for the following reasons. It provides intuitive control in task space, making it easy to specify actions in task space. It enables generalization across different robot types since the control signals are independent of the robot's internal configuration. It allows for scalability, as applying an OSC per arm enables generalization to an arbitrary number of robot arms.

The OSC implementation in robosuite follows standard operational space control theory, extending Equation (2.1) to

$$\boldsymbol{\tau} = \underbrace{\mathbf{J}^T(\mathbf{q})\mathbf{u}}_{\boldsymbol{\tau}_{\text{task}}} + \boldsymbol{\tau}_{\text{null}} + \boldsymbol{\tau}_{\text{g}}$$

by incorporating gravity compensation $\boldsymbol{\tau}_{\text{g}}$ to enhance end-effector accuracy and null-space control $\boldsymbol{\tau}_{\text{null}}$. The latter guides the robot around problematic singularities and keeps it in a configuration of high manipulability [75].

The primary control signal \mathbf{u} for task-space movements is based on a PD controller [4], ensuring compliant behavior:

$$\mathbf{u} = \mathbf{K}_P(\mathbf{x}_{\text{des}} - \mathbf{x}) + \mathbf{K}_D(\dot{\mathbf{x}}_{\text{des}} - \dot{\mathbf{x}}) .$$

Here \mathbf{x}_{des} and \mathbf{x} are the desired and current end-effector pose. $\dot{\mathbf{x}}_{\text{des}}$ and $\dot{\mathbf{x}}$ are the corresponding velocities. Stiffness \mathbf{K}_P and damping \mathbf{K}_D define the compliance and damping behavior of the controller. For this work, the default settings of ROBOSUITE are used to implement a critical damping: $\mathbf{K}_P = 150 \cdot \mathbf{I}_6$ and $\mathbf{K}_D = 2\sqrt{\mathbf{K}_P}$. The reader should note that the units for positional and rotational entries vary, but the given values follow the International System of Units (SI).

To simplify learning, the desired velocity is set to zero in all cases. This suffices for manipulation tasks and reduces the action space dimensionality. Additionally, ROBOSUITE provides implementations for *absolute* and *delta* control inputs. This work uses *delta* inputs since they are independent of the task-space size. With this choice, the final control formulation is

$$\boldsymbol{\tau} = \mathbf{J}^T(\mathbf{q})(\mathbf{K}_P\Delta\mathbf{x} - \mathbf{K}_D\dot{\mathbf{x}}) + \boldsymbol{\tau}_{\text{null}} + \boldsymbol{\tau}_{\text{g}} .$$

The controller's input is $\Delta\mathbf{x} \in \mathbb{R}^6$. The first three components correspond to the desired change in position $\Delta\mathbf{p}$. The last three components describe the desired change in rotation $\Delta\mathbf{r}$, expressed using the axis-angle representation.

4.1.2. Gripper Control

The gripper control is a simple proportional controller for each joint q of the gripper:

$$\tau_q = k_p(q_{\text{des}} - q).$$

The ROBOSUITE's default uses a rigid setup with $k_p = 1000$. The SI unit again depends on the joint type.

4.2. Action Space

Like the controller, the action space is decomposed into control signals for each end-effector and each gripper.

4.2.1. End-Effector Action Space

For a single arm, the end-effector action space for the end-effector is defined as $\mathbf{a} \in [-1, 1]^6$. The action values are normalized to the range $[-1, 1]$ and are then mapped to the actual controller ranges of $\Delta\mathbf{x}$, which in this work falls to ± 5 cm for position and ± 0.5 rad for rotation.

4.2.2. Gripper Action Space

The gripper action space is $[-1, 1]$. The lower limit -1 corresponds to fully opening the gripper, and $+1$ corresponds to fully closing the gripper. This gripper action is independent of the number of fingers and synchronizes all of them. This is sufficient for tasks that require grasping simplistic objects, but depending on the task and its objects, this can be a limiting factor. Furthermore, it should be noted that the change of the gripper state is limited to ± 0.2 per time step: To change the command signal from fully open to fully closed or visa versa, at least $2/0.2 = 10$ time steps are needed.

For internal control, the input actions are then internally scaled to the corresponding joint limits of the individual fingers.

4.2.3. Total Action Space

Combining the above components, the total action space for N_a arms becomes $[-1, 1]^{7N_a}$. Each arm contributes seven dimensions, six for the end-effector and one for the gripper:

$$\mathbf{a} = \begin{pmatrix} \Delta \tilde{\mathbf{p}} \\ \Delta \tilde{\mathbf{r}} \\ \tilde{q}_{\text{grip}} \end{pmatrix} \in [-1, 1]^7 \quad (4.1)$$

The tilde indicates that these actions are scaled to $[-1, 1]$. A scaling function s is used to transfer between the scaled values and the actual input values for the controller parts:

$$\tilde{\bullet} = s(\bullet) \text{ and } \bullet = s^{-1}(\tilde{\bullet}), \text{ for } \bullet = \Delta \mathbf{p}, \Delta \mathbf{r}, q_{\text{grip}}. \quad (4.2)$$

In tasks without a gripper, the action space dimension is reduced accordingly. The actions are queried with a frequency of 20 Hz, while the underlying MuJoCo simulation runs with 500 Hz.

4.3. Observation Space

ROBOSUITE provides a highly flexible observation space, supporting joint positions and velocities, poses in the world frame, RGB images, RGB-D images, and segmentation masks. Arbitrary other observation modalities can be added.

5. Bimanual Tasks

A core goal of this work is to provide a structured set of bimanual tasks that cover a wide range of coordination challenges. The tasks are categorized based on the arms' temporal, spatial, and physical coupling and the requirement for symmetrical and synchronized coordination, following the categorization proposed in [18]. Additionally, a horizon category indicates the approximate duration needed to complete the task. Furthermore, it is ensured that all selected tasks actually require two arms. ROBOSUITE [63] already provides four two-arm tasks; another four were added to achieve sufficient task coverage. For specific task details, the author is referred to Appendix A.

5.1. Existing Bimanual Tasks

The existing bimanual tasks in the ROBOSUITE framework are illustrated in Figure 5.1. The Figure highlights the flexibility of different robot configurations in ROBOSUITE.

The *peg-in-hole* task requires the robot to guide a cylindrical stick through a square hole. This task does not involve grippers, as the objects are rigidly attached to the robot arms. The coupling in this scenario is primarily spatial, as both arms must work within a confined area while exerting force to achieve proper insertion. Physical and temporal coupling does not appear. Each arm contributes equally to the insertion process without a strict timing constraint. The coordination required is asymmetric but synchronized.

The *lift* task is a short-horizon task in which both arms must grasp and lift an edgy cup from a table. The cup is designed to be easily gripped, minimizing the complexity of the grasping process. This task exhibits spatial and physical coupling, as the arms must operate within the same workspace and apply force simultaneously. Coordination in this task is symmetric and synchronized since both arms must exert equal force and lift the object in a coordinated manner to maintain balance and prevent unintended tilting.

The *handover* task requires one robot arm to pick up a hammer and transfer it to the

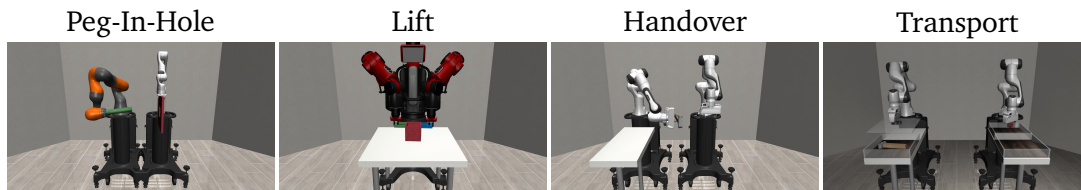


Figure 5.1.: Bimanual Tasks of ROBOSUITE [63] with varying robot configurations. *Peg-in-hole*: The green pole has to be stuck through the hole of the red plate. *Lift*: The two arms must lift the squared cub. *Handover*: One arm has to pick up the hammer from the table and hand it over to the other arm. *Transport*: The red cube must be moved from the rear box to the box up front. The lid of the left box must be removed, and the hammer must be placed in the rear box. Here, it can be seen how lightning and the environment’s appearance can be adapted.

other arm. This task introduces temporal coupling, as the handover movement follows a sequential sequence. The task exhibits a mix of spatial and physical coupling, as both arms must interact with the same object in a controlled manner. In this case, the required coordination is asymmetric but synchronized. A simplified version of this task exists, where the hammer starts in the gripper of one of the arms, reducing the complexity. The *transport* task introduces an additional layer of complexity, requiring a hammer to be moved from an initial bin to a target bin while simultaneously removing an obstructing object from the target bin. An extra challenge is introduced by covering the target bin with a lid, which must be lifted before placing the hammer inside. This task exhibits all three types of coupling: temporal, as the lifting of the lid and transport of the hammer must be executed sequentially; spatial, as both arms operate in overlapping workspaces; and physical, during the handover. The coordination is asymmetric but again synchronized.

5.2. New Bimanual Tasks

Four additional tasks have been implemented to extend the range of bimanual challenges beyond the existing ROBOSUITE tasks. They are depicted in Figure 5.2. These tasks focus on increasing difficulty in coupling, coordination, and the necessity for precise manipulation.

The *place-ball* task involves lifting and placing a large ball inside a box. The ball is designed to be too large for a single-arm grasp, necessitating the use of both arms for lifting. The

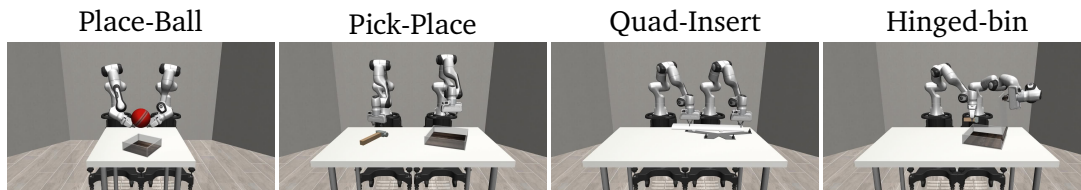


Figure 5.2.: Implemented tasks in the bimanual framework. *Place-ball*: The two arms must lift and place the ball in a box up front. *Pick-place*: Including a handover, the hammer must be picked up and placed in the box. *Quad-insert*: The bracket must be placed correctly on the four pins with 1 mm tolerance. *Hinged-bin*: The hammer must be placed in the hinged bin. The opening angle of the lid is limited so that the box does not stay open by itself. At the end of the task, the bin must be closed.

task demonstrates strong spatial coupling, as the arms must coordinate their forces and maintain a stable ball hold. Coordination is symmetric and synchronized, requiring the arms to move together to avoid dropping the ball.

The *pick-place* task extends the complexity of the handover task by requiring the hammer to be placed inside a small box after transfer. This increases the temporal horizon and introduces additional precision constraints. The task mainly exhibits temporal and spatial coupling, as the arms must work in sequence to complete the task. The coordination is asymmetric and synchronized, with one arm passing the object and the other executing the final placement.

The *quad-insertion task*, adapted from [17], presents a significantly more challenging scenario compared to the *lift* task. The two arms must grasp a bracket with two handles, move it to the target location, and precisely insert it onto four pins with a tight tolerance of 1 mm. This task requires exceptional spatial and physical coupling, as both arms must exert precise control to achieve alignment. Coordination is symmetric and synchronized, as the arms must simultaneously adjust their positioning to ensure successful insertion. Finally, the *hinged-bin* task involves picking up a hammer and placing it inside a bin that is closed with a hinged lid. One arm is responsible for lifting the lid, while the other places the hammer inside. Unlike more straightforward object-placement tasks, like the *put-bottle-in-fridge* [11], this scenario mandates actual bimanual coordination, as the lid has a limited opening angle and does not remain open once released. The task exhibits strong temporal and spatial coupling, as both arms must operate synchronously to complete the sequence. Coordination is asymmetric and synchronized, with each arm having distinct but interdependent roles.









Task	Coupling			Coordination		Horizon
	temporal	spatial	physical	symmetrical	synchronized	
Peg-In-Hole	✗	✓	✗	✗	✓	
Lift	✗	✓	✓	✓	✓	
Handover	✓	✓	✓	✗	✓	
Transport	✓	✓	✓	✗	✓	
Place-Ball	✗	✓	✗	✓	✓	
Pick-Place	✓	✓	✓	✗	✓	
Quad-Insert	✗	✓	✓	✓	✓	
Hinged-Bin	✓	✓	✗	✓	✓	

Table 5.1.: Task categorization of the bimanual tasks following [18]. Besides the occurring/required coupling and coordination, the task horizon is indicated.

Together with these new tasks, the framework’s tasks, summarized in Table 5.1, ensure a broad range of coupling, coordination, and task horizon challenges. The reader should note that all presented tasks require synchronized coordination and have spatial coupling. This highlights the actual need for two arms to solve the challenges.

5.3. Randomization and Task Variations

Each task incorporates randomization to enhance robustness in policy learning. The initial robot joint configurations, object positions, and orientations are randomized. Specific object properties, such as the size of the hammer, also vary to increase generalization. The exact details of the randomization setup can be found in Appendix A.

With its code-first approach, ROBOSUITE enables a straightforward adaption of the task’s appearances and allows the generation of various task variants by, for example, adapting the distribution for sampling the initial states.

6. Data Collection

Collecting high-quality demonstration data is essential for IL, and numerous methods have been developed to acquire such trajectories. ROBOTURK [62], for example, relies on mobile phone-based teleoperation, primarily designed for single-arm tasks. MIMICGEN [56] amplifies a small number of expert demonstrations by generating additional synthetic trajectories. D3IL [57] utilizes an Xbox gamepad for teleoperation, focusing on single-arm setups. ALOHA [10] employs an expensive teleoperation system, while VITAL [76] introduces a low-cost teleoperation approach, both aimed explicitly at bimanual tasks. These methods highlight the range of existing approaches to the demonstration collection.

The ROBOSUITE framework [63] itself provides several fundamental pre-implemented data collection methods, including keyboard-based control, GUI-based operation, and the use of the SpaceMouse[®] by 3DCONNEXION. However, these approaches are mostly suited for simple one-handed tasks. The SpaceMouse[®] device requires extensive training before effective teleoperation.

The data collection and generation process is not the primary focus of this work. Instead, the approach of [17], [18] is adapted, leveraging waypoint-based experts for data collection. This method enables fast and reproducible generation of thousands of expert trajectories, ensuring high-quality and consistent training data. Nevertheless, it should be mentioned that ROBOSUITE provides an interface to add new teleoperation devices and methods.

6.1. Waypoint-Based Expert

Following the work of [17], [18], an expert policy based on waypoints is utilized. The waypoints for both arms are jointly specified as $(\mathbf{t}_{\text{left}}, \mathbf{t}_{\text{right}})$, where each waypoint is characterized by a target position \mathbf{p}_{WP} , orientation \mathbf{r}_{WP} , and gripper state \tilde{q}_{WP} , denoted

as $\mathbf{t} = (\mathbf{p}_{\text{WP}}, \mathbf{r}_{\text{WP}}, \tilde{q}_{\text{grip,WP}})$. The desired gripper state takes a value of either -1 (open) or 1 (closed).

The waypoint targets are dynamically computed based on the observations following the randomized environment reset. For tasks requiring high accuracy, such as *quad-insertion*, waypoints can be dynamically updated after each action step based on the current observations. This mechanism also enables the waypoint-based expert to work in potential dynamic environments.

For a single arm in state $\mathbf{s} = (\mathbf{p}_{\text{current}}, \mathbf{r}_{\text{current}})$, the corresponding *delta* action is computed using a P-controller-like approach based on the current waypoint target \mathbf{t} :

$$\mathbf{a}(\mathbf{s}, \mathbf{t}) = \begin{pmatrix} s(\mathbf{p}_{\text{WP}} - \mathbf{p}_{\text{current}}) \\ s \left(\text{Log} \left(\text{Exp}(\mathbf{r}_{\text{WP}}) \text{Exp}(\mathbf{r}_{\text{current}})^T \right) \right) \\ \tilde{q}_{\text{grip,WP}} \end{pmatrix} \in [-1, 1]^6 \times \{-1, 1\}. \quad (6.1)$$

The scaling function s (Equation (4.2)) ensures that the action values remain within the predefined input range of $[-1, 1]$. The term $\text{Log} \left(\text{Exp}(\mathbf{r}_{\text{WP}}) \text{Exp}(\mathbf{r}_{\text{current}})^T \right)$ computes the rotation delta $\Delta \mathbf{r}$ in an axis-angle representation. Unlike the end-effector actions, the gripper action is applied immediately without interpolation. The overall execution of the waypoint expert within the environment is detailed in Algorithm 6.1.

6.2. Data Storage and Processing

The expert demonstration data is stored efficiently as pairs of simulation states and corresponding actions $\mathcal{D}_s = \{\mathbf{s}_i, \mathbf{a}_i\}_i$. This memory-efficient approach enables post-collection adaptation of observation modalities, e.g., switching between low-dimensional state-based observations, RGB(D) images, or segmentation data. Before initiating the learning, it is beneficial to generate an explicit dataset, as later discussed in Section 7.2. This ensures optimized and efficient data accessibility during the models' training.

```

1: state = environment.reset()                                ▷ Get initial state.
2: waypoints = create_waypoints(state)
3: success = false
4: for each waypoint in waypoints do
5:     reached = false                                        ▷ Track if waypoint is reached.
6:     while not reached do
7:         if waypoint.is_dynamic then
8:             waypoint = update(waypoint, state)            ▷ Update waypoint.
9:         end if
10:        action = get_action(waypoint, state)              ▷ Equation (6.1)
11:        state, success = environment.step(action)         ▷ Execute action.
12:        reached = check_reached(waypoint, state)         ▷ Check if waypoint is reached.
13:        if success then
14:            terminate
15:        end if
16:    end while
17: end for

```

Algorithm 6.1.: Rollout of the waypoint expert for a single episode. The code is a simplified version of the implementation, without data collection, and checks for, e.g., reachability and tolerances.

7. Learning Framework

With `ROBOSUITE` [63] serving as the foundational environment for simulation and data collection, the choice of `ROBOMIMIC` [54] for the learning component is a natural one, given that both are part of the Advancing Robot Intelligence through Simulated Environments (ARISE) initiative [77]. Since these modules originate from the same development group, their compatibility is well-established, ensuring seamless integration.

`ROBOMIMIC` provides a modular and well-structured framework for training, evaluation, and rollout of policies. It includes training pipelines for various IL and offline RL methods: Different variants of BC [7], Hierarchical BC [78], and offline RL algorithms such as TD3-BC [79], IQL [80], IRIS [81] are implemented. Beyond these built-in methods, `ROBOMIMIC` offers a well-structured base for implementing custom methods and algorithms, making it a suitable choice for extending learning strategies.

7.1. Algorithm Overview

`ROBOMIMIC` [54] provides several pre-implemented algorithms for IL. The framework supports multiple BC variants, including standard BC [7] and versions that utilize stochastic Gaussian Mixture Model policies [82], Variational Autoencoders [83], Recurrent Neural Networks [84], and Transformer-based policies [32]. Furthermore, Hierarchical BC [78] is implemented to target long-horizon tasks.

In addition to these pre-existing methods, this work extends the framework by incorporating additional algorithms based on generative models: Diffusion Policy [9], Flow Matching [34], ActionFlow [1], and a variant of ActionFlow without the invariance property. Further details on the methodology and implementation of these algorithms are provided in Chapter 8.

7.2. Learning Pipeline

By combining ROBOSUITE [63] and ROBOMIMIC [54], this work establishes a structured pipeline for a full IL loop. The suggested workflow consists of the following steps:

1. Create a ROBOSUITE task or select an existing one, e.g. Section 5.
2. Define a waypoint-based expert or choose an existing one (Chapter 6). Alternatively, use an existing data collection method from ROBOSUITE or introduce a new teleoperation device.
3. Collect expert demonstrations using ROBOSUITE, resulting in a dataset of simulation state-action pairs:

$$\mathcal{D}_s = \{(\mathbf{s}_i, \mathbf{a}_i)\} .$$

4. Determine the environment appearance and observation modalities (low-dimensional, RGB, depth, segmentation) before extracting the observation-action pairs from \mathcal{D}_s to obtain the learning dataset:

$$\mathcal{D}_o = \{(\mathbf{o}_i, \mathbf{a}_i)\} .$$

5. Execute the training, evaluation, and rollout of policies using ROBOMIMIC.

This pipeline offers a modular and flexible framework for imitation learning, allowing seamless integration of new tasks, data collection strategies, and learning algorithms. Its structured design ensures reproducibility and scalability, making it an effective tool for conducting the experiments detailed in Chapter 9.

8. Algorithms

In this chapter, three algorithms are presented: Diffusion Policy [9], Flow Matching [34], and ActionFlow [1]. The Diffusion Policy represents a SOTA approach for imitation learning. While its structure is not specifically tailored to robotic tasks, it has demonstrated strong performance across various domains. Before this work, Diffusion Policy had not been integrated into ROBOMIMIC [54]. Its integration into the framework provides a solid benchmark against which ActionFlow can be tested. The same applies to Flow Matching. Incorporating ActionFlow into the learning framework, it is also extended to support multiple robotic arms N_a . This work focuses on the bimanual case of $N_a = 2$. The following sections provide a detailed theoretical background on the algorithms and discuss the modifications introduced for the implementation in this work. Furthermore, a slight ActionFlow modification is proposed, replacing the Invariant Point Attention (IPA) with a standard Multi-Head Attention (MHA). This model will be referred to as *noIPA*. It adapts the flow mechanism to pose-based tasks and, from a concept perspective, sits between Flow Matching and ActionFlow.

8.1. Common Policy Structure

All methods are implemented as multi-step policies. This means that instead of predicting only the next action based on the current observation, the model considers a sequence of past observations to generate a sequence of future actions. This approach provides a more stable and informed decision-making process. Formally, the last T_o observations are used to predict the next T_p actions, of which only the first T_a are executed before a new prediction is made. For clarity, the following notations are used throughout this chapter: The observation sequence of length T_o is denoted as $\mathbf{o} = (\mathbf{o}_{t-T_o+1}, \dots, \mathbf{o}_{t-1}, \mathbf{o}_t)$ and the predicted action sequence of length T_p as $\mathbf{a} = (\mathbf{a}_t, \mathbf{a}_{t+1}, \dots, \mathbf{a}_{t+T_p-1})$.

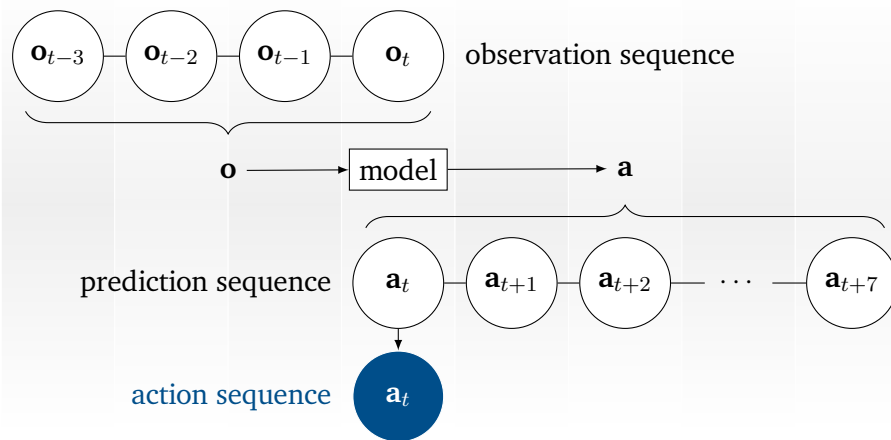


Figure 8.1.: The multi-step inference cycle at timestep t . Based on the latest T_o observations, the next T_p actions are predicted, but only the first T_a actions are executed. After that, the inference starts again for timestep $t + T_a$. The scheme uses the values $T_o = 4$, $T_p = 8$ and $T_a = 1$.

During training, the model learns from (\mathbf{o}, \mathbf{a}) pairs, effectively loosening the Markov property [6] by incorporating temporal dependencies: The model predicts an action sequence based on the latest observation history at inference time but only executes the first T_a actions before updating the prediction. This process is illustrated in Figure 8.1. A history of observations allows the model to estimate velocities and infer temporal patterns. Predicting a sequence of actions instead of a single action improves trajectory smoothness and helps the model commit to a single mode of behavior. Additionally, executing multiple actions per inference step ($T_a > 1$) reduces overall inference time and can aid in meeting real-time requirements.

For the experiments in Chapter 9, the values $T_o = 4$, $T_p = 8$, and $T_a = 1$ are used. Optimizing these hyperparameters was not the focus of this work, and depending on the task and setup, alternative configurations may yield better performance.

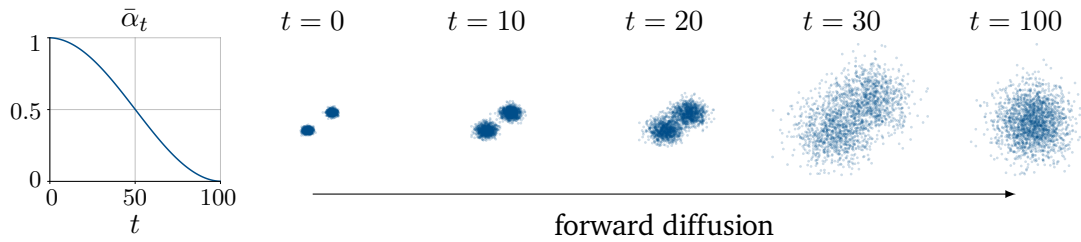


Figure 8.2.: Forward diffusion process $\mathbf{x}^{(t)} = \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)} + \sqrt{1 - \bar{\alpha}_t}\epsilon$ in 2D with a cosine schedule. The initial *blob*-distribution ($t = 0$) is washed out until only Gaussian noise remains ($t = 100$). A model is trained to reverse this process.

8.2. Diffusion Policy

The Diffusion Policy [9] is a SOTA model that applies diffusion models to IL. It builds upon the principles of diffusion models [36], [85], [86], initially developed for generative modeling, and adapts them to predict actions. Unlike BC, which directly maps observations to actions, Diffusion Policy learns to model the distribution of plausible actions by iteratively refining a noisy initial guess.

8.2.1. Diffusion Process for Action Generation

The core idea behind diffusion models is to gradually corrupt training data with Gaussian noise and then train a neural network to recover the original data step by step. The fundamental idea of the diffusion process is sketched in the following; for mathematical details, the reader is referred to [36], [87].

The forward diffusion process iteratively adds noise to a sample $\mathbf{a}^{(0)}$ from the true distribution, forming a Markov chain of increasingly noisy samples. Mathematically, this process can be expressed as

$$q\left(\mathbf{a}^{(t)}|\mathbf{a}^{(t-1)}\right) = \mathcal{N}\left(\mathbf{a}^{(t)}; \sqrt{1 - \beta_t}\mathbf{a}^{(t-1)}, \beta_t\mathbf{I}\right),$$

where β_t is the noise variance schedule: $\{\beta_t \in (0, 1)\}_{t=1}^T$ [36]. For $T \rightarrow \infty$, $\mathbf{a}^{(T)}$ is equivalent to a sample from the zero mean Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Figure 8.2 illustrates the diffusion process.

Using the parametrization trick with $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$ the noise-infused sample at step t can be computed in closed-form as

$$\begin{aligned} \mathbf{a}^{(t)} &= \sqrt{\alpha_t} \mathbf{a}^{(t-1)} + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon} \\ &= \sqrt{\alpha_t \alpha_{t+1}} \mathbf{a}^{(t-2)} + \sqrt{1 - \alpha_t \alpha_{t-1}} \boldsymbol{\epsilon} \\ &= \dots = \sqrt{\bar{\alpha}_t} \mathbf{a}^{(0)} + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}. \end{aligned} \quad (8.1)$$

As $\boldsymbol{\epsilon}$ is noise sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, this implements

$$q(\mathbf{a}^{(t)} | \mathbf{a}^{(0)}) = \mathcal{N}(\mathbf{a}^{(t)}; \sqrt{\bar{\alpha}_t} \mathbf{a}^{(0)}, 1 - \bar{\alpha}_t \mathbf{I}).$$

The reverse process aims to denoise the input starting at a random sample $\mathbf{a}^{(T)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. During training a model p_θ is learned to predict the noise removal:

$$\mathcal{L}_\theta = \left\| \boldsymbol{\epsilon} - p_\theta(\mathbf{a}^{(t)}, \mathbf{o}, t) \right\|^2. \quad (8.2)$$

The denoising model $\hat{\boldsymbol{\epsilon}} = p_\theta(\mathbf{a}^{(t)}, \mathbf{o}, t)$ is conditioned on the current observation sequence \mathbf{o} and the diffusion step t .

8.2.2. Learning and Inference

During training, the model is trained to predict $\boldsymbol{\epsilon}$. To achieve this, the model is trained on observation-state $(\mathbf{o}, \mathbf{a}) \in \mathcal{D}_{\text{train}}$ to minimize the loss in Equation (8.2). Algorithm 8.1 depicts this.

Equation 8.1 suggests that one could directly predict $\mathbf{a}^{(0)}$ from $\mathbf{a}^{(T)}$ using $\hat{\boldsymbol{\epsilon}} = p_\theta(\mathbf{a}^{(T)}, \mathbf{o}, T)$ and $\mathbf{a}^{(0)} = (\mathbf{a}^{(T)} - \sqrt{1 - \bar{\alpha}_T} \hat{\boldsymbol{\epsilon}}) / \sqrt{\bar{\alpha}_T}$. However, this direct computation does not align with the generative nature of diffusion. The reverse diffusion process is designed to gradually remove noise over multiple steps, ensuring a smooth and high-quality reconstruction. Therefore, inference instead follows a stepwise procedure where at each step, an estimate $\mathbf{a}^{(t-1)}$ is computed from $\mathbf{a}^{(t)}$. Additionally, a controlled noise $\sigma_t \mathbf{z}$ is introduced to maintain the probabilistic structure of the generative process [88]:

$$\mathbf{a}^{(t-1)} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{a}^{(t)} - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \hat{\boldsymbol{\epsilon}} \right) + \sigma_t \mathbf{z}, \quad \text{with } \sigma_t^2 = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} (1 - \alpha_t) \text{ and } \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}).$$

- 1: $t \sim \text{Uniform}(\{1, \dots, T\})$
- 2: $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- 3: $\mathbf{a}^{(t)} = \sqrt{\bar{\alpha}_t} \mathbf{a} + \sqrt{1 - \bar{\alpha}_t} \epsilon \quad \triangleright \text{Eq. (8.1)}$
- 4: $\hat{\epsilon} = \mathbf{p}_\theta(\mathbf{a}^{(t)}, \mathbf{o}, t)$
- 5: $\mathcal{L}_\theta = \|\epsilon - \hat{\epsilon}\|^2$

- 1: $\hat{\mathbf{a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \triangleright \text{Gaussian noise } \mathbf{a}^{(T)}$
- 2: **for** $t = t_N, \dots, t_1$ **do**
- 3: $\hat{\epsilon} = \mathbf{p}_\theta(\hat{\mathbf{a}}, \mathbf{o}, t) \quad \triangleright \text{Predict noise.}$
- 4: Update $\hat{\mathbf{a}}$ using Equation (8.3)
- 5: **end for**
- 6: **return** $\hat{\mathbf{a}} \quad \triangleright \text{Generated action } \hat{\mathbf{a}}^{(0)}$

Algorithm 8.1.: Diffusion training step for a single observation-action pair $(\mathbf{o}, \mathbf{a}) \in \mathcal{D}_{\text{train}}$. The computed loss \mathcal{L}_θ is used for mini-batch updates of the model \mathbf{p}_θ .

Algorithm 8.2.: Diffusion inference step for a given observation \mathbf{o} and N inference timesteps $\{t_1, \dots, t_N\} \subset \{1, \dots, T\}$ for DDIM inference [87].

The authors of [87] introduce a deterministic formulation of the reverse process that allows mapping $\mathbf{a}^{(t)}$ to any earlier timestep $\tau < t$:

$$\mathbf{a}^{(\tau)} = \underbrace{\sqrt{\bar{\alpha}_\tau} \left(\frac{\overbrace{\mathbf{a}^{(t)} - \sqrt{1 - \bar{\alpha}_t} \hat{\epsilon}}^{\text{Eq. (8.1) with predicted } \mathbf{a}^{(0)}}}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{predicted } \mathbf{a}^{(0)}} + \sqrt{1 - \bar{\alpha}_\tau} \hat{\epsilon}, \quad \text{with } \hat{\epsilon} = \mathbf{p}_\theta(\mathbf{a}^{(t)}, \mathbf{o}, t). \quad (8.3)$$

This decreases the inference time, as fewer steps can be used while maintaining a good sampling quality. Algorithm 8.2 summarizes the inference cycle.

8.2.3. Transformer Architecture

This work uses a Transformer-based model \mathbf{p}_θ . The Transformer model [32] is a SOTA approach that has demonstrated strong performance across various domains beyond natural language processing.

Transformers have revolutionized deep learning by introducing attention mechanisms that efficiently process long-range dependencies. The architecture for action prediction differs from standard implementations in its tokenization approach. Instead of text embeddings, the input consists of an observation- and action-sequence (\mathbf{o} and \mathbf{a}) and a time index indicating the current step in the diffusion process. Implementing [9], the modalities are projected into a latent space of dimensionality D_{emb} before being processed by the

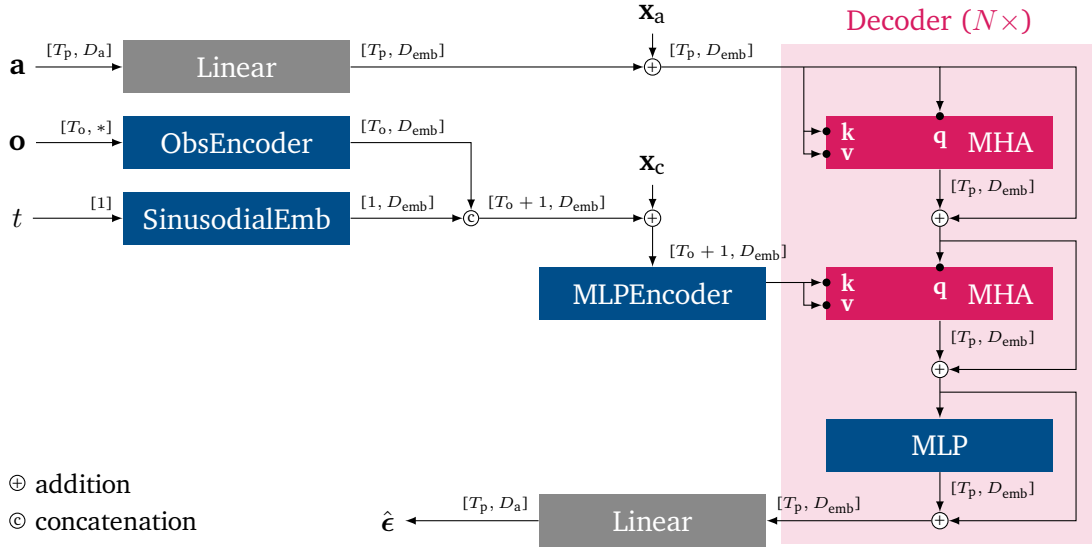


Figure 8.3.: The diffusion model $p_{\theta}(\mathbf{a}, \mathbf{o}, t)$ following [9]. Normalization and dropout layers are not shown. The action sequence \mathbf{a} , the observation sequence \mathbf{o} , and the diffusion timestep t are embedded into a latent space of dimensionality D_{emb} . The position parameters $\mathbf{x}_{\mathbf{a}/\mathbf{c}}$ are learnable. In total $T_p + T_o + 1$ tokens are encoded. After N decoder iterations with Multi-Head Attention (MHA), a final linear layer recovers the action dimensionality D_a . The first MHA block implements a self-attention block.

attention mechanism; Figure 8.3 illustrates the model. For simplicity, normalization and dropout layers are excluded in the following explanation.

The action sequence \mathbf{a} of length T_p is passed through a single linear layer and is combined with a learnable position parameter $\mathbf{x}_{\mathbf{a}}$.

The observation sequence \mathbf{o} of length T_o is passed through an observation encoder. Depending on the available observation modalities, Multilayer perceptrons (MLPs), Convolutional Neural Networks (CNNs), or other components are used.

The timestep parameter t is embedded using a sinusoidal encoding.

Concatenating the encoded observations and timestep, the total conditioning receives a learnable position parameter $\mathbf{x}_{\mathbf{c}}$ and passes through a final MLP encoder.

As a result, the encoded action sequence has T_p tokens, and the conditioning has $T_o + 1$ tokens. After passing through N decoder layers, a final linear layer restores the action dimensionality.

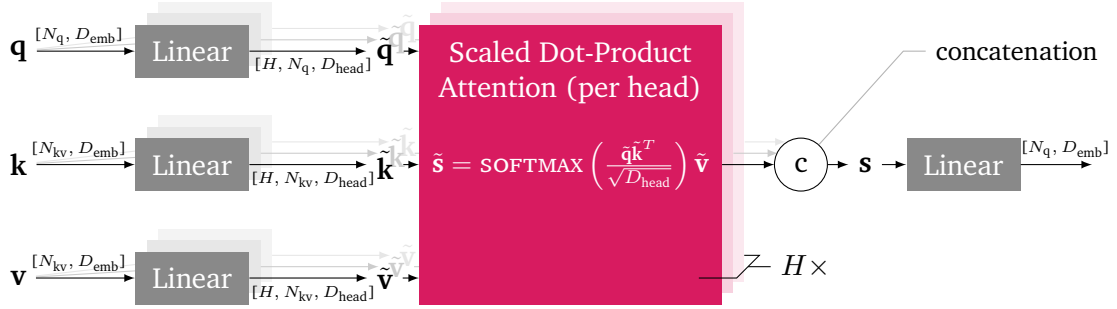


Figure 8.4.: Multi-Head Attention implementation of [32]. The inputs \mathbf{q} , \mathbf{k} and \mathbf{v} , with $N_{q/kv}$ tokens each, are projected to H heads. Per head, an attention score $\tilde{\mathbf{s}}$ is computed. These scores are concatenated to \mathbf{s} . A final linear layer forms the output. Inside the `SOFTMAX` of the score calculation, attention masks \mathbf{M} can be applied.

Multi-Head Attention. The heart of the described transformer model is the Multi-Head Attention (MHA) block [32]. All details about it can be found in [32], but the fundamentals are displayed here to later compare it to the attention mechanism of the ActionFlow [1]. The MHA block maps a query \mathbf{q} and key-value pairs (\mathbf{k}, \mathbf{v}) to an output. Figure 8.4 illustrates this: Each of the three input vectors is passed through a linear layer projecting them to H heads of dimensionality $D_{\text{head}} = D_{\text{emb}}/H$, resulting in $\tilde{\mathbf{q}}$, $\tilde{\mathbf{k}}$ and $\tilde{\mathbf{v}}$. For each head, an attention score

$$\tilde{\mathbf{s}} = \text{SOFTMAX} \left(\frac{\tilde{\mathbf{q}}\tilde{\mathbf{k}}^T}{\sqrt{D_{\text{head}}}} \right) \tilde{\mathbf{v}} \quad (8.4)$$

is computed using Scaled Dot-Product Attention. Details for Equation (8.4) are given in a moment. These attention scores are then concatenated again to create a final score value \mathbf{s} with the same shape as \mathbf{a} . Passing \mathbf{s} through a final linear layer creates the output (with the same shape as \mathbf{a}).

In the model \mathbf{p}_θ from above, this attention mechanism is used twice in the decoder: In the first self-attention block, each decoder token can attend to each other decoder token, but causal masking ensures that tokens can only attend to previous tokens. This is achieved using an additional mask matrix \mathbf{M} inside the `SOFTMAX` of Equation (8.4). In the second MHA, each decoder token can attend to all tokens from the encoding. Here, an attention mask \mathbf{M} is used to prevent attention to padded positions in the encoder sequence, ensuring that the model does not incorporate information from padding tokens, which do not carry meaningful content.

Scaled Dot-Product Attention. Equation (8.4) describes the Scaled Dot-Product Attention used in the MHA block. The underlying computations are explained here, as this is later needed to understand the theory behind the ActionFlow model. Regarding a single head, the inputs to the attention functions are the query $\tilde{\mathbf{q}} \in \mathbb{R}^{N_q \times D}$ and the key-value pair $\tilde{\mathbf{k}} \in \mathbb{R}^{N_{kv} \times D}$, $\tilde{\mathbf{v}} \in \mathbb{R}^{N_{kv} \times D}$. The head dimensionality D_{head} is abbreviated as D . The query is noted as

$$\tilde{\mathbf{q}} = \begin{pmatrix} q_{11} & \cdots & q_{1D} \\ \vdots & & \vdots \\ q_{N_q 1} & \cdots & q_{N_q D} \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_{N_q}^T \end{pmatrix}.$$

The vectors $\mathbf{q}_i \in \mathbb{R}^D$ describe the query part of token $i \in \{1, \dots, N_q\}$. Analogously, key and value are written as

$$\tilde{\mathbf{k}} = \begin{pmatrix} \mathbf{k}_1^T \\ \vdots \\ \mathbf{k}_{N_{kv}}^T \end{pmatrix} \text{ and } \tilde{\mathbf{v}} = \begin{pmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{N_{kv}}^T \end{pmatrix}.$$

The dot-product $\tilde{\mathbf{q}}\tilde{\mathbf{k}}^T$ computes the importance score of each key \mathbf{k}_j for every query \mathbf{q}_i per row:

$$\tilde{\mathbf{q}}\tilde{\mathbf{k}}^T = \begin{pmatrix} \mathbf{q}_1^T \\ \vdots \\ \mathbf{q}_{N_q}^T \end{pmatrix} \begin{pmatrix} \mathbf{k}_1 \cdots \mathbf{k}_{N_{kv}} \end{pmatrix} = \begin{pmatrix} \mathbf{q}_1^T \mathbf{k}_1 & \cdots & \mathbf{q}_1^T \mathbf{k}_{N_{kv}} \\ \vdots & & \vdots \\ \mathbf{q}_{N_q}^T \mathbf{k}_1 & \cdots & \mathbf{q}_{N_q}^T \mathbf{k}_{N_{kv}} \end{pmatrix} \in \mathbb{R}^{N_q \times N_{kv}}.$$

$(\tilde{\mathbf{q}}\tilde{\mathbf{k}}^T)_{ij} \in \mathbb{R}$ is interpreted as the importance of key j for query i . To normalize the importance per query, each row is normalized by the `SOFTMAX` function, such that $\sum_j (\tilde{\mathbf{q}}\tilde{\mathbf{k}}^T)_{ij} = 1 \forall i$. The scaling with \sqrt{D} is applied to prevent vanishing gradients. The final score of the head is computed as

$$\tilde{\mathbf{s}} = \begin{pmatrix} \mathbf{q}_1^T \mathbf{k}_1 & \cdots & \mathbf{q}_1^T \mathbf{k}_{N_{kv}} \\ \vdots & & \vdots \\ \mathbf{q}_{N_q}^T \mathbf{k}_1 & \cdots & \mathbf{q}_{N_q}^T \mathbf{k}_{N_{kv}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_{N_{kv}}^T \end{pmatrix} = \begin{pmatrix} \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_{N_q} \end{pmatrix} \in \mathbb{R}^{N_q \times D}.$$

A single cell

$$s_{ij} = \mathbf{q}_i^T \mathbf{k}_1 v_{1j} + \dots + \mathbf{q}_i^T \mathbf{k}_{N_{kv}} v_{N_{kv}j}$$

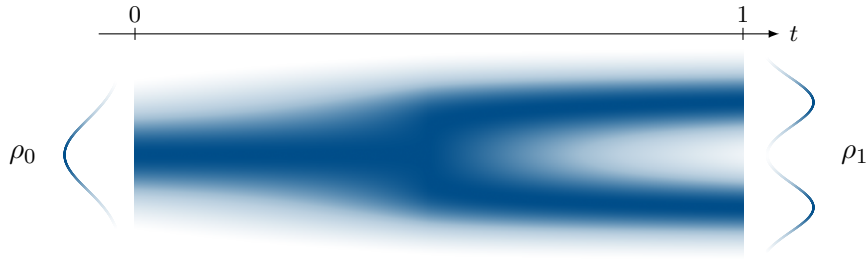


Figure 8.5.: Linear probability flow ρ_t from a normal Gaussian $\rho_0 = \mathcal{N}(0, 1)$ to a *blob* distribution ρ_1 . The visualization is derived from [89]. The darker the color, the higher the probability density.

corresponds to the interaction of one query \mathbf{q}_i with all keys $\tilde{\mathbf{k}}$ and all values $\tilde{\mathbf{v}}$.

8.3. Flow Matching

Unlike diffusion-based methods, which iteratively refine noisy samples through denoising, Flow Matching [34] models a probability flow between a source and target distribution. This section presents the mathematical principles behind Flow Matching and the learning and inference mechanism. The underlying Transformer architecture is equivalent to the one presented in the previous Section 8.2.

8.3.1. Flow Matching for Action Generation

Flow Matching [34] is a continuous time formulation for modeling probability distributions. Instead of progressively corrupting data with noise and learning to reverse this process (diffusion), Flow Matching directly models the probability flow between the source and target distributions using an ordinary differential equation. Let $\rho_1(\mathbf{a})$ be the data distribution over actions and $\rho_0(\mathbf{a})$ be a reference distribution, often chosen as a simple prior such as a Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The probability path $\rho_t(\mathbf{a})$ for $t \in [0, 1]$ connects these two distributions and implies the motion of $\mathbf{a}^{(0)} \sim \rho_0(\mathbf{a})$ to its target $\mathbf{a}^{(1)} \sim \rho_1(\mathbf{a})$ as the flow $\mathbf{a}^{(t)} = \phi_t(\mathbf{a}^{(0)})$. Instead of modeling ϕ_t directly, Flow Matching proposes to learn the vector field $\mathbf{u}_t(\mathbf{a}) = d\phi_t(\mathbf{a})/dt$ with a parametric model $\mathbf{v}_\theta(\mathbf{a}, t)$ [90]. Learning this directly is generally intractable, as no unique and closed-form solution for \mathbf{u}_t

exists. This is solved by conditioning the vector field and, therefore, the flow on the target distribution. The choice for the target flow is chosen to be a straight line from $\mathbf{a}^{(0)}$ to $\mathbf{a}^{(1)}$:

$$\mathbf{a}^{(t)} = \phi_t(\mathbf{a}^{(0)}, \mathbf{a}^{(1)}) = t\mathbf{a}^{(1)} + (1-t)\mathbf{a}^{(0)}, \quad \mathbf{u}_t = \frac{d\phi_t}{dt} = \mathbf{a}^{(1)} - \mathbf{a}^{(0)}. \quad (8.5)$$

This induces a constant and tractable vector field $\mathbf{u} = \mathbf{u}_t$. Figure 8.5 visualizes the probability path ρ_t for this linear interpolation

Adding the observations \mathbf{o} as conditioning the objective of Flow Matching becomes

$$\mathcal{L}_\theta = \left\| \mathbf{u}_t(\mathbf{a}, \mathbf{a}^{(1)}) - \mathbf{v}_\theta(\mathbf{a}, \mathbf{o}, t) \right\|^2. \quad (8.6)$$

In contrast to the Diffusion model [36], the model is not predicting noise removal but learns to predict the velocity of the underlying flow to iteratively update noisy samples $\mathbf{a}^{(0)}$ until $\mathbf{a}^{(1)}$ is reached. Typically, this is done by applying Euler integration

$$\mathbf{a}^{(t_{k+1})} = \mathbf{a}^{(t_k)} + \mathbf{v}_\theta(\mathbf{a}^{(t_k)}, \mathbf{o}, t_k) \Delta t, \quad \Delta t = t_{k+1} - t_k, \quad (8.7)$$

for K explicit times $t_k \in [0, 1]$ with $t_0 = 0$ and $t_K = 1$.

8.3.2. Learning and Inference

The principal structure of the training and inference with the Flow Matching [34] looks similar to that of the Diffusion Policy (Section 8.2.2). Still, instead of learning to predict a noise ϵ , the model \mathbf{v}_θ is trained to predict the vector field \mathbf{u} of the flow ϕ . The diffusion process is replaced with the flow mechanism of Equation (8.5). The training objective is similar to that of the Diffusion Policy but pushes the model to learn the correct velocity (Equation (8.6)) instead of the noise removal. A linear schedule for t is used, e.g. $t \in \mathcal{T}_{\text{train}} = \{0, 0.01, 0.02, \dots, 1.0\}$ for 100 training steps. During inference, Euler integration is applied to update the action prediction (Equation (8.7)) by following the vector field of the learned flow. During inference, the predicted velocities move the action sequence gradually along the probability path. Similar to using DDIM [87] in the Diffusion Policy, it is possible to use an exponential schedule during inference with Flow Matching: Starting with more significant steps, the step width exponentially decreases for t closer to 1. Figure 8.6 visualizes this exponential subset $\mathcal{T}_{\text{inference}} \subset \mathcal{T}_{\text{train}}$.

Algorithms 8.3 and 8.4 clarify the model structure and application. Compared to the Algorithms 8.1 and 8.2, it shows that the structure of training inference is equal to that of the Diffusion Policy. Only the diffusion process has been replaced by flow matching.

```

1:  $t \sim \text{Uniform}(\mathcal{T}_{\text{train}})$ 
2:  $\mathbf{a}^{(0)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
3:  $\mathbf{u} = \mathbf{a} - \mathbf{a}^{(0)}$ 
4:  $\mathbf{a}^{(t)} = t\mathbf{a} + (1-t)\mathbf{a}^{(0)}$ 
5:  $\hat{\mathbf{u}} = \mathbf{v}_{\theta}(\mathbf{a}^{(t)}, \mathbf{o}, t)$ 
6:  $\mathcal{L}_{\theta} = \|\mathbf{u} - \hat{\mathbf{u}}\|^2$ 

```

Algorithm 8.3.: Flow Matching training step of the Flow Matching model for a single observation-action pair $(\mathbf{o}, \mathbf{a}) \in \mathcal{D}_{\text{train}}$. The computed loss \mathcal{L}_{θ} is used for mini-batch updates of the model v_{θ} .

```

1:  $\hat{\mathbf{a}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
2: for  $t \in \mathcal{T}_{\text{inference}} \setminus \{0\}$  do
3:    $\hat{\mathbf{u}} = \mathbf{v}_{\theta}(\hat{\mathbf{a}}, \mathbf{o}, t)$ 
4:    $\hat{\mathbf{a}} \leftarrow \hat{\mathbf{a}} + \hat{\mathbf{u}}\Delta t$ 
5: end for
6: return  $\hat{\mathbf{a}}$ 

```

Algorithm 8.4.: Inference step with the Flow Matching model for a given observation \mathbf{o} , with an exponential schedule $\mathcal{T}_{\text{inference}}$.

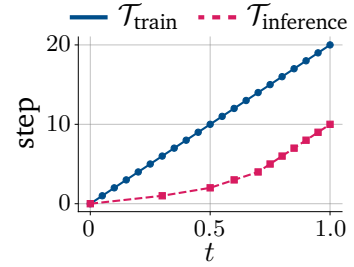


Figure 8.6.: Linear and exponential schedules for Flow Matching. The exponential schedule enables faster inference. Here, 20 training and the 10 corresponding inference steps are visualized.

The model $\mathbf{v}_{\theta}(\mathbf{a}, \mathbf{o}, t)$ is equivalent to the model $\mathbf{p}_{\theta}(\mathbf{a}, \mathbf{o}, t)$ used in the Diffusion Policy (Figure 8.3), but instead of predicting the noise removal ϵ , the model returns the estimated velocity \mathbf{u} of the flow ϕ . As before, T_{p} action tokens and $T_{\text{o}} + 1$ tokens for conditioning are used in the MHA block.

8.4. ActionFlow

ActionFlow [1] introduces a pose-based approach to imitation learning for robotic tasks and employs Flow Matching ([34] and Section 8.3) as its core generative mechanism. ActionFlow represents each pose as distinct tokens within an adapted Transformer architecture. While traditional Transformers [32] originate from text processing, where each token corresponds to a word, ActionFlow adapts this concept by assigning a unique token to each pose in a manipulation scene. This structured tokenization provides an intuitive way to model interactions within the environment, reinforcing spatial dependencies. Further, MHA is replaced with Invariant Point Attention (IPA), making the model invariant to global pose transformations. Before the specialized pose-based Transformer architecture for Flow Matching is introduced, the action and observation space of ActionFlow is clarified.

8.4.1. Action and Observation Space Adaption

The environment (Chapter 4) and the previously presented methods utilize *delta* actions \mathbf{a} and vectorized observations \mathbf{o} . In contrast, ActionFlow [1] relies on a pose-based scene and action representation. A pose is defined as $\mathbf{T} = (\mathbf{p}, \mathbf{R}) \in \mathbb{R}^3 \times \text{SO}(3)$, where \mathbf{p} represents the position and \mathbf{R} the orientation in 3D space.

Each action prediction for a single end-effector consists of a target pose and an additional feature:

$$\mathbf{A}_t = (\mathbf{T}_{a,t}, \mathbf{f}_{a,t}) .$$

The feature \mathbf{f} can store additional action-related information, such as the target gripper state \tilde{q}_{grip} . Since the current end-effector position and orientation are known, it is always possible to map between pose-based actions \mathbf{A} and the *delta* actions \mathbf{a} .

Similarly, each observation consists of a pose and associated features:

$$\mathbf{O}_t = (\mathbf{T}_{o,t}, \mathbf{f}_{o,t}) .$$

Potential observation features include the gripper state of the observed end-effector, RGB images from wrist cameras, or point clouds of observed objects. These observation tuples can be extracted from the environment observations \mathbf{o} .

Throughout this work, all poses \mathbf{T} are assumed to be represented in the same coordinate frame, such as the world frame, ensuring a consistent spatial representation. Considering the multi-step policy structure, the action and observation spaces are formulated as follows:

$${}^{(a)}\mathbf{A} = \left({}^{(a)}\mathbf{A}_t, {}^{(a)}\mathbf{A}_{t+1}, \dots, {}^{(a)}\mathbf{A}_{t+T_p-1} \right) \text{ and } {}^{(o)}\mathbf{O} = \left({}^{(o)}\mathbf{O}_{t-T_o+1}, \dots, {}^{(o)}\mathbf{O}_{t-1}, {}^{(o)}\mathbf{O}_t \right) .$$

Here, T_p and T_o represent the prediction and observation horizons, respectively. The notation $a = 1, \dots, N_a$ denotes individual end-effectors, while $o = 1, \dots, N_o$ indicates each observed pose in the scene. With

$$\mathbf{A} = \left({}^{(1)}\mathbf{A}, \dots, {}^{(N_a)}\mathbf{A} \right) , \quad \mathbf{O} = \left({}^{(1)}\mathbf{O}, \dots, {}^{(N_o)}\mathbf{O} \right) ,$$

the action and observation spaces consist of pose-feature tuples structured of shape (N, T) , where N denotes the number of elements, and T is the temporal horizon.

For example, with a prediction horizon of $T_p = 8$ and $N_a = 2$ robot arms, the action space consists of $T_p N_a = 16$ pose-feature pairs. With $N_o = 3$ observed poses and an observation horizon of $T_o = 4$, the observation space has $N_o T_o = 12$ pose-feature pairs.

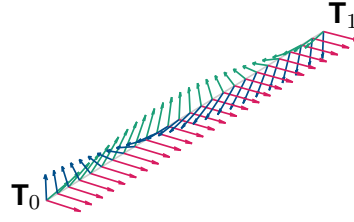


Figure 8.7.: Flow in $\mathbb{R}^3 \times \text{SO}(3)$ from $\mathbf{p}_0 = \mathbf{0}$, $\mathbf{R}_0 = \mathbf{I}$ to $\mathbf{p}_1 = (1 \ 1 \ 1)^T$, $\mathbf{R}_1 = (\mathbf{e}_x - \mathbf{e}_y - \mathbf{e}_z)$ with x , y and z axes in red, green and blue. The displayed frames correspond to the times $t = 0, 0.04, 0.08, \dots, 1$ of the flow.

8.4.2. Flow for Poses

Implementing the flow for poses $\mathbf{T} = (\mathbf{p}, \mathbf{R})$ demands extra attention, as interpolation for rotations is not straightforward. The following elaboration shows how the flow from the ActionFlow paper [1], which mainly follows [91], is implemented and simplified. For better readability, the indice $t \in [0, 1]$ of the flow mechanism is moved from top $\mathbf{T}^{(t)}$ to the bottom \mathbf{T}_t . This is not to be confused with the time index in the action and observation sequences. The flow mechanism is discussed for a single pose \mathbf{T} but can be multiplied by parallel computation for all relevant poses.

Following [91], [92], \mathbb{R}^3 and $\text{SO}(3)$ are modeled independently. The flow in $\mathbb{R}^3 \times \text{SO}(3)$ moves an initial pose $\mathbf{T}_0 = (\mathbf{p}_0, \mathbf{R}_0)$ to a target pose $\mathbf{T}_1 = (\mathbf{p}_1, \mathbf{R}_1)$ on a straight line.

$$\begin{aligned} \mathbf{p}_t &= \phi_t(\mathbf{p}_0 | \mathbf{p}_1) = t\mathbf{p}_1 + (1-t)\mathbf{p}_0, \\ \mathbf{R}_t &= \phi_t(\mathbf{R}_0 | \mathbf{R}_1) = \mathbf{R}_0 \text{Exp}(t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1)), \end{aligned} \quad (8.8)$$

for $t \in [0, 1]$. Log and Exp are the logarithmic and the exponential map of the $\text{SO}(3)$ manifold [93]. Figure 8.7 displays a visualization of this flow.

The flow in Equation (8.8) is decoupled. Therefore, the vector field $\mathbf{u}_t = d\phi_t/dt$ is decoupled, and the translational and rotational components can be considered independently.

Translational Component. The velocity $\dot{\mathbf{p}}_t$ that is induced by the flow can be computed via

$$\dot{\mathbf{p}}_t = \frac{d\phi_t}{dt} = \mathbf{p}_1 - \mathbf{p}_0. \quad (8.9)$$

Note that this velocity is constant for all times t . This is also expected, as the flow is linear in time. The same is implemented in [1], [91]: The velocity is approximated with

$$\dot{\mathbf{p}}_t = \frac{\mathbf{p}_1 - \mathbf{p}_t}{1 - t},$$

where \mathbf{p}_t is computed via $\mathbf{p}_t = t\mathbf{p}_1 + (1 - t)\mathbf{p}_0$. This is equal to Equation (8.9):

$$\dot{\mathbf{p}}_t = \frac{\mathbf{p}_1 - \mathbf{p}_t}{1 - t} = \frac{\mathbf{p}_1 - t\mathbf{p}_1 - (1 - t)\mathbf{p}_0}{1 - t} = \frac{(1 - t)\mathbf{p}_1 - (1 - t)\mathbf{p}_0}{1 - t} = \mathbf{p}_1 - \mathbf{p}_0.$$

As the velocity is constant over time, it will be denoted as $\mathbf{u}_p = \dot{\mathbf{p}}$ in the following.

Rotational Component. The time derivative of the rotational flow component is

$$\dot{\mathbf{R}}_t = \frac{d\phi_t}{dt} = \underbrace{\mathbf{R}_0 \text{Exp}(t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1))}_{=\mathbf{R}_t} \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1) = \mathbf{R}_t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1).$$

Here, $d/dt(\text{Exp}(t\mathbf{A})) = \mathbf{A}\text{Exp}(t\mathbf{A})$ is utilized. With $\dot{\mathbf{R}} = \mathbf{R}\dot{\mathbf{r}}$ [93] the velocity can be identified as

$$\dot{\mathbf{r}}_t = \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1). \quad (8.10)$$

It should be noted that this velocity is relative to the local coordinate frame. In [1], [91] the velocity is approximated with

$$\dot{\mathbf{r}}_t = \frac{\text{Log}(\mathbf{R}_t^{-1} \mathbf{R}_1)}{1 - t}. \quad (8.11)$$

Analyzing $\mathbf{R}_t^{-1} \mathbf{R}_1$ gives the following:

$$\begin{aligned} \mathbf{R}_t^{-1} \mathbf{R}_1 &\stackrel{(8.8)}{=} [\mathbf{R}_0 \text{Exp}(t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1))]^{-1} \mathbf{R}_1 = [\text{Exp}(t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1))]^{-1} \mathbf{R}_0^{-1} \mathbf{R}_1 \\ &= \text{Exp}\left(t \text{Log}\left([\mathbf{R}_0^{-1} \mathbf{R}_1]^{-1}\right)\right) \mathbf{R}_0^{-1} \mathbf{R}_1 \stackrel{(i)}{=} \text{Exp}(-t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1)) \text{Exp}(\text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1)) \\ &\stackrel{(ii)}{=} \text{Exp}(-t \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1) + \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1)) = \text{Exp}((1 - t) \text{Log}(\mathbf{R}_0^{-1} \mathbf{R}_1)) \end{aligned} \quad (8.12)$$

with (i) : $\text{Log}(\mathbf{A}^{-1}) = -\text{Log}(\mathbf{A})$ and (ii) : $\text{Exp}(\mathbf{a})\text{Exp}(\mathbf{b}) = \text{Exp}(\mathbf{a} + \mathbf{b})$.

The reader should note, that (ii) generally violates the Baker-Campbell-Hausdorff formula [94] for non-commuting \mathbf{a} and \mathbf{b} , but holds in this special case with $\mathbf{a} = -t\mathbf{b}$, resulting in

$$\mathbf{ab} - \mathbf{ba} = -t\mathbf{bb} - \mathbf{b}(-t\mathbf{b}) = 0.$$

Inserting Equation (8.12) this into Equation (8.11) results in

$$\begin{aligned} \dot{\mathbf{r}}_t &= \frac{\text{Log}(\mathbf{R}_t^{-1}\mathbf{R}_1)}{1-t} \stackrel{(8.12)}{=} \frac{\text{Log}(\text{Exp}((1-t)\text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1)))}{1-t} \\ &= \frac{(1-t)\text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1)}{1-t} = \text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1). \end{aligned}$$

This shows that the velocities of the Equations (8.10) and (8.11) are identical. Due to the time independence, this velocity will be denoted as $\mathbf{u}_r = \dot{\mathbf{r}}$.

Total velocity component. As mentioned above, the rotational velocity is expressed in the local frame. The translational velocity should also be represented in the local frame. This is achieved by pre-multiplying with \mathbf{R}_t^T [1]:

$$\begin{aligned} \mathbf{u}_p &= \underbrace{\mathbf{R}_t^T (\mathbf{p}_1 - \mathbf{p}_0)}_{\text{local frame}}^{\text{world frame}} \\ \mathbf{u}_r &= \underbrace{\text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1)}_{\text{local frame}}. \end{aligned} \tag{8.13}$$

As a result, the translational velocity component is time-dependent again. Deciding for the local frame, instead of the global frame, is also crucial to later achieve invariance to global transformations, as shown in the ActionFlow paper [1].

8.4.3. Learning and Inference

With the flow being adapted to be pose-specific, the learning and inference have to be adapted, too. Thus, the model is structured to output the velocity components for translation and position, but the prediction for the gripper target is absolute. In other words, the flow mechanism is only applied to the pose part of the action. Algorithm 8.5 summarizes a learning step for an observation-action pair $(\mathbf{O}, \mathbf{A}) \in \mathcal{D}_{\text{train}}$, the actions position, rotation and gripper sequences are denoted as $(\mathbf{p}, \mathbf{R}, \tilde{q}_{\text{grip}})$. Similar to Section 8.3.2 a linear time schedule $\mathcal{T}_{\text{train}}$ is used.

-
- 1: $\mathbf{p}, \mathbf{R}, \tilde{q}_{\text{grip}} \leftarrow \mathbf{A}$ ▷ Extract trajectory components.
 - 2: $t \sim \text{Uniform}(\mathcal{T}_{\text{train}})$ ▷ $t \in [0, 1]$
 - 3: $\mathbf{p}_0, \mathbf{R}_0 \sim \text{RANDN}(\mathbb{R}^3, \text{SO}(3))$
 - 4: $\mathbf{p}_1, \mathbf{R}_1 \leftarrow \mathbf{p}, \mathbf{R}$
 - 5: Compute flow at time t :

$$\mathbf{p}_t = t\mathbf{p}_1 + (1-t)\mathbf{p}_0, \quad \mathbf{R}_t = \mathbf{R}_0 \text{Exp}(t \text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1)), \quad \mathbf{A}_t = (\mathbf{p}_t, \mathbf{R}_t)$$

- 6: Compute vector field at time t in local frame:

$$\mathbf{u}_p = \mathbf{R}_t^T (\mathbf{p}_1 - \mathbf{p}_0) \quad \mathbf{u}_r = \text{Log}(\mathbf{R}_0^{-1}\mathbf{R}_1)$$

- 7: Run model prediction:

$$\hat{\mathbf{u}}_p, \hat{\mathbf{u}}_r, \hat{q}_{\text{grip}} = \mathbf{v}_\theta(\mathbf{A}_t, \mathbf{O}, t)$$

- 8: Compute loss:

$$\mathcal{L}_\theta = w_p \|\mathbf{u}_p - \hat{\mathbf{u}}_p\|^2 + w_r \|\mathbf{u}_r - \hat{\mathbf{u}}_r\|^2 + w_{\text{grip}} \|\tilde{q}_{\text{grip}} - \hat{q}_p\|^2 \quad (8.14)$$

Algorithm 8.5.: Training step for a pose based flow model \mathbf{v}_θ on a single end-effector action $(\mathbf{p}, \mathbf{R}, \tilde{q}_{\text{grip}})$ and observation \mathbf{O} . $\mathcal{T}_{\text{train}}$ contains linearly spaced flow times t from 0 and 1. The computed loss \mathcal{L}_θ is used for mini-batch updates of the model and consists of weighted components $(w_{p/r/grip})$ of the translational, rotational, and gripper loss.

-
- 1: $\hat{\mathbf{A}} = (\hat{\mathbf{p}}, \hat{\mathbf{R}}) \sim \text{RANDN}(\mathbb{R}^3, \text{SO}(3))$
 - 2: **for** $t \in \mathcal{T}_{\text{inference}} \setminus \{0\}$ **do**
 - 3: $\hat{\mathbf{u}}_p, \hat{\mathbf{u}}_r, \hat{q}_{\text{grip}} = \mathbf{v}_\theta(\hat{\mathbf{A}}, \mathbf{O}, t)$
 - 4: Update $\hat{\mathbf{A}}$ following Equation (8.15):

$$\hat{\mathbf{p}} \leftarrow \hat{\mathbf{p}} + \hat{\mathbf{R}}\hat{\mathbf{u}}_p\Delta t, \quad \hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}}\text{Exp}(\hat{\mathbf{u}}_r\Delta t).$$

- 5: **end for**
- 6: **return** $\hat{\mathbf{A}}, \hat{q}_{\text{grip}}$

Algorithm 8.6.: Inference step for a pose-based flow model, given an observation \mathbf{O} . To speed up inference time, an exponential subset of discrete time steps $\mathcal{T}_{\text{inference}}$ is used. The flow mechanism is only applied to position and rotation.

During inference, the predicted velocities move the poses gradually. An exponential $\mathcal{T}_{\text{inference}} \subset \mathcal{T}_{\text{train}}$ schedule is used to speed up inference (see Figure 8.6). Using the models' predicted velocities $\hat{\mathbf{u}}_{\text{p}}$ and $\hat{\mathbf{u}}_{\text{r}}$, the action sequence is successively updated using Euler integration:

$$\mathbf{p}_{t_{k+1}} = \mathbf{p}_{t_k} + \mathbf{R}_{t_k} \hat{\mathbf{u}}_{\text{p}} \Delta t, \quad \mathbf{R}_{t_{k+1}} = \mathbf{R}_{t_k} \text{Exp}(\hat{\mathbf{u}}_{\text{r}} \Delta t), \quad \Delta t = t_{k+1} - t_k. \quad (8.15)$$

The translational velocity is moved from the local to the world frame by pre-multiplying with \mathbf{R} . A complete inference step is displayed in Algorithm 8.6.

8.4.4. Invariant Transformer

The underlying model $\mathbf{v}_{\theta}(\mathbf{A}, \mathbf{O}, t)$ of the ActionFlow policy is based on a transformer encoder, but instead of using Multi-Head Attention (MHA), Invariant Point Attention (IPA) [35] is used. Figure 8.8 summarizes the model structure. To implement this, the features $\mathbf{f}_{\mathbf{A}}$, $\mathbf{f}_{\mathbf{O}}$ of the candidate action sequence \mathbf{A} and the observations \mathbf{O} run through individual encoders. This maps these features to a common embedding dimensionality D_{emb} . If no features are available for a pose, the encoder output is a vector of zeros. Fitting encoder models, like MLPs or CNNs, are used depending on the incoming feature modality. The embedded features are concatenated along the token dimensionality. Similarly, all poses are listed along this dimensionality. For N_{a} arms, N_{o} observed poses, a prediction horizon T_{p} , and an observation horizon T_{o} a total of

$$N_{\text{t}} = N_{\text{a}}T_{\text{p}} + N_{\text{o}}T_{\text{p}}$$

tokens of pose feature pairs $(\mathbf{p}, \mathbf{R}, \mathbf{f})$ are created. The features of these poses are combined with a learnable position parameter $\mathbf{x} \in \mathbb{R}^{N_{\text{t}} \times D_{\text{emb}}}$. t is embedded using sinusoidal embedding, too, and a MLP. In contrast to the Diffusion Policy, this embedding is not used to create another token but is replicated N_{t} times before it is added to the features \mathbf{f} . The transformer encoder consists of N encoder blocks built using IPA and a MLP. The outputs are updated tokens of shape $\mathbb{R}^{N_{\text{t}} \times D_{\text{emb}}}$. From these tokens, the first $N_{\text{a}}T_{\text{p}}$ action tokens are passed through a final linear layer. This maps each output token to \mathbb{R}^7 , with the first six entries being the predicted velocities $\hat{\mathbf{u}}_{\text{p}}$ and $\hat{\mathbf{u}}_{\text{r}}$, and the last entry corresponding to the gripper state \hat{q}_{gripper} .

Comparing this model to the models of Diffusion Policy and Flow Matching (Figure 8.3 on page 30) shows the key differences: While Diffusion Policy and Flow Matching uses one token per sequence element and one additional token to embed t ($T_{\text{p}} + T_{\text{o}} + 1$ tokens in

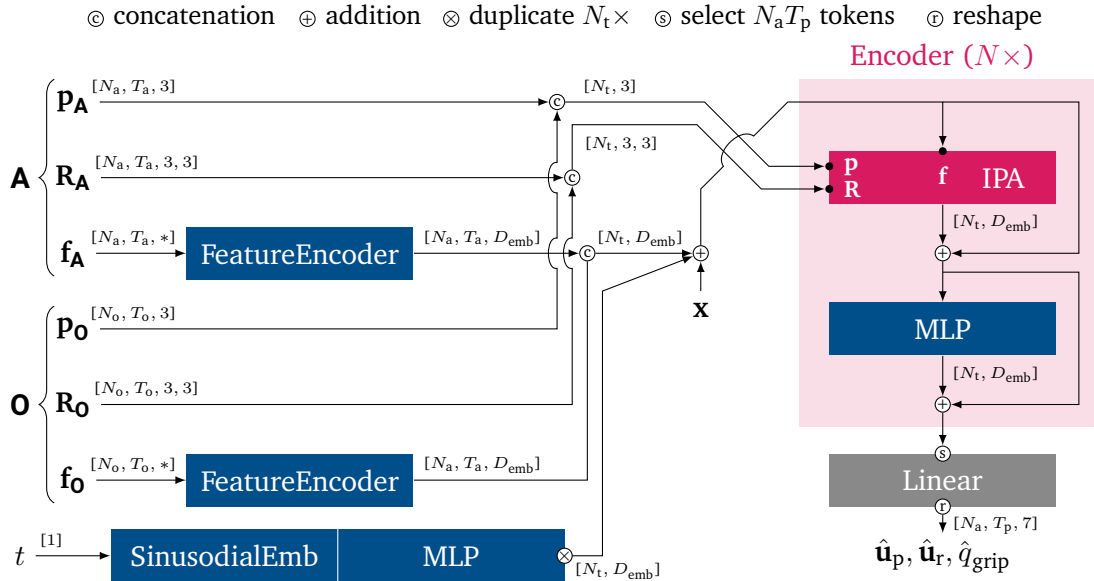


Figure 8.8.: The ActionFlow model $\mathbf{v}_\theta(\mathbf{A}, \mathbf{O}, t)$ adapted from [1]. Normalization and dropout layers are not shown. The features \mathbf{f} of the action sequence \mathbf{A} and the observation sequence \mathbf{O} , as well as the flow step t , are embedded into a space of dimensionality D_{emb} . \mathbf{x} is a learnable position parameter. In total $N_t = N_a T_p + N_o T_p$ tokens are encoded. After N transformer iterations with Invariant Point Attention (IPA), a final linear layer recovers the output dimensionality \mathbb{R}^7 .

total), ActionFlow uses one token per pose ($N_a T_p + N_o T_p$ in total). Moreover, ActionFlow combines each embedded feature token with a pose (\mathbf{p}, \mathbf{R}) that is used in the IPA, which is used instead of MHA.

Invariant Point Attention. The key component of ActionFlow is the IPA [35]. This section describes the simplified IPA implementation that is used in ActionFlow [1]. In addition to processing features in the latent space, IPA incorporates geometric information by utilizing the corresponding positions $\mathbf{r} \in \mathbb{R}^3$ and rotations $\mathbf{R} \in \text{SE}(3)$. Given that, the conventional attention is extended by incorporating explicit geometric transformations. The details are presented in the following. As the IPA is used as self-attention, the number of tokens N_t and the head dimensionality D_{head} are abbreviated with N and D .

Projection and Multi-Head Splitting. Similar to self-attention with MHA, linear layers project the feature tokens $\mathbf{f} \in \mathbb{R}^{N_t \times D_{\text{emb}}}$ to queries \mathbf{q} , keys \mathbf{k} , and values \mathbf{v}

$$(\mathbf{q}, \mathbf{k}, \mathbf{v}) = \text{LINEAR}(\mathbf{f}),$$

which are then split into H heads:

$$\tilde{\mathbf{q}}, \tilde{\mathbf{k}}, \tilde{\mathbf{v}} \in \mathbb{R}^{H \times N_t \times D_{\text{head}}}.$$

This is equivalent to the initial embedding of MHA; see Figure 8.4.

Incorporating Pose Information. Each token is associated with pose information $\mathbf{T} = (\mathbf{p}, \mathbf{R})$ given by a rotation matrix $\mathbf{R} \in \text{SO}(3)$ and a translation vector $\mathbf{p} \in \mathbb{R}^3$. This information is consolidated in a transformation matrix \mathbf{T} and its inverse:

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix} \in \text{SE}(3), \text{ and } \mathbf{T}^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T \mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix} \in \text{SE}(3).$$

Pose-Invariant Attention Score. The attention score of MHA (Equation 8.4 and page 32) is modified to include the pose information \mathbf{T} . Given the fact that self-attention is applied to the queries $\tilde{\mathbf{q}}$ and key-value pairs $\tilde{\mathbf{k}}, \tilde{\mathbf{v}}$ of a single head are elements in $\mathbb{R}^{N \times D}$. Using the pose information $\mathbf{T}_i, i = \{1, \dots, N\}$ of the individual tokens and following the notation introduced on page 32, each vector of these three components is transformed by

$$\mathbf{q}'_i = (\mathbf{T}_i^{-1})^T \mathbf{q}_i, \mathbf{k}'_i = \mathbf{T}_i \mathbf{k}_i, \text{ and } \mathbf{v}'_i = \mathbf{T}_i \mathbf{v}_i.$$

Typically, the head dimensionality D is larger than 4. Thus, these transformations are implemented as

$$\mathbf{Ax} = \begin{pmatrix} \mathbf{Ax}_{[1:4]} \\ \mathbf{Ax}_{[5:8]} \\ \vdots \\ \mathbf{Ax}_{[D-3:D]} \end{pmatrix} \text{ for } \mathbf{A} \in \mathbb{R}^{4 \times 4} \text{ and } \mathbf{x} \in \mathbb{R}^D.$$

In total, the transformed components are

$$\tilde{\mathbf{q}}' = \begin{pmatrix} \mathbf{q}_1^T \mathbf{T}_1^{-1} \\ \vdots \\ \mathbf{q}_N^T \mathbf{T}_N^{-1} \end{pmatrix}, \quad \tilde{\mathbf{k}}' = \begin{pmatrix} (\mathbf{T}_1 \mathbf{k}_1)^T \\ \vdots \\ (\mathbf{T}_1 \mathbf{k}_N)^T \end{pmatrix}, \quad \text{and } \tilde{\mathbf{v}}' = \begin{pmatrix} (\mathbf{T}_1 \mathbf{v}_1)^T \\ \vdots \\ (\mathbf{T}_1 \mathbf{v}_N)^T \end{pmatrix}.$$

The attention function of MHA from Equation (8.4) is used on dashed components:

$$\tilde{\mathbf{s}}' = \text{SOFTMAX} \left(\frac{\tilde{\mathbf{q}}' \tilde{\mathbf{k}}'^T}{\sqrt{D_{\text{head}}}} \right) \tilde{\mathbf{v}}'.$$

Looking into the details of the dot product reveals its invariance:

$$\tilde{\mathbf{q}}' \tilde{\mathbf{k}}'^T = \begin{pmatrix} \mathbf{q}_1^T \mathbf{T}_1^{-1} \mathbf{T}_1 \mathbf{k}_1 & \cdots & \mathbf{q}_1^T \mathbf{T}_1^{-1} \mathbf{T}_N \mathbf{k}_N \\ \vdots & & \vdots \\ \mathbf{q}_N^T \mathbf{T}_N^{-1} \mathbf{T}_1 \mathbf{k}_1 & \cdots & \mathbf{q}_N^T \mathbf{T}_N^{-1} \mathbf{T}_N \mathbf{k}_{N_{kv}} \end{pmatrix}$$

The importance of key \mathbf{k}_j for query \mathbf{q}_i is extended to contain the relative transformation between the poses \mathbf{T}_i and \mathbf{T}_j :

$$\left(\tilde{\mathbf{q}}' \tilde{\mathbf{k}}'^T \right)_{ij} = \mathbf{q}_i^T \overbrace{\mathbf{T}_i^{-1} \mathbf{T}_j}^{\text{relative transformation } {}^i\mathbf{T}_j} \mathbf{k}_j.$$

This importance is invariant to any global transformations $\mathbf{T}_\delta \in \text{SE}(3)$ as

$$(\mathbf{T}_\delta \mathbf{T}_i)^{-1} (\mathbf{T}_\delta \mathbf{T}_j) = \mathbf{T}_i^{-1} \underbrace{\mathbf{T}_\delta^{-1} \mathbf{T}_\delta}_{=\mathbf{I}} \mathbf{T}_j = \mathbf{T}_i^{-1} \mathbf{T}_j.$$

This proves that the dot products are computed solely based on the relative transformations between tokens. Similar to the standard Scaled-Dot-Product Attention on page 32, the importance is normalized per row. Using the updated dot product, the final attention score is computed as

$$\tilde{\mathbf{s}}' = \left(\tilde{\mathbf{q}}' \tilde{\mathbf{k}}'^T \right) \tilde{\mathbf{v}}' = \begin{pmatrix} \mathbf{q}_1^T \mathbf{T}_1^{-1} \mathbf{T}_1 \mathbf{k}_1 & \cdots & \mathbf{q}_1^T \mathbf{T}_1^{-1} \mathbf{T}_N \mathbf{k}_N \\ \vdots & & \vdots \\ \mathbf{q}_N^T \mathbf{T}_N^{-1} \mathbf{T}_1 \mathbf{k}_1 & \cdots & \mathbf{q}_N^T \mathbf{T}_N^{-1} \mathbf{T}_N \mathbf{k}_{N_{kv}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1^T \mathbf{T}_1^T \\ \vdots \\ \mathbf{v}_N^T \mathbf{T}_N^T \end{pmatrix}.$$

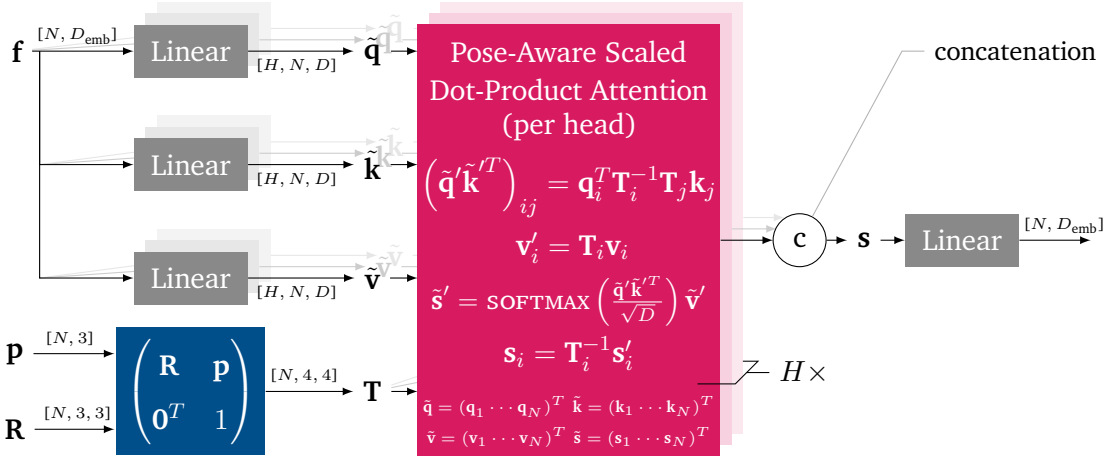


Figure 8.9.: The full IPA block of ActionFlow [1]. The structure is similar to the one of the MHA block in Figure 8.4 but incorporates the pose information \mathbf{T} .

The values \mathbf{v}_i have been mapped to the frame of the keys \mathbf{k}_i . However, the attention output is expected to be in the same frame as its input. Thus, the final output of the pose-aware attention function is mapped back to the input frame: $\mathbf{s}_i = (\mathbf{T}_i^{-1} \mathbf{s}'_i)^T$. Given that, the total attention function is invariant to global transformations \mathbf{T}_δ :

$$\text{IPA}(\mathbf{f}, \mathbf{T}) = \text{IPA}(\mathbf{f}, \mathbf{T}_\delta \mathbf{T}).$$

The output of the attention function is processed through a final linear layer, like in the MHA block. Figure 8.9 summarizes the pose-invariant attention.

8.4.5. Invariance of ActionFlow

With the IPA being invariant to global transformations \mathbf{T}_δ , the velocity model is invariant too because the IPA component is the first and only element processing the poses. Mathematically this is expressed by

$$\mathbf{v}_\theta(\mathbf{A}, \mathbf{O}, t) = \mathbf{v}_\theta(\mathbf{T}_\delta \odot \mathbf{A}, \mathbf{T}_\delta \odot \mathbf{O}, t).$$

Here \odot marks that the global transformation \mathbf{T} is applied to all poses \mathbf{T} in the action and observation sequences: $\mathbf{T}' = \mathbf{T}_\delta \mathbf{T}$. With this underlying invariant backbone and the fact that the target velocities for Flow Matching are given in the local frame, see

Equation (8.13), the policy is invariant to global transformations, as shown by Funk *et al.* [1] in detail. This property suggests that ActionFlow is well-suited for tasks involving spatially structured inputs, such as those found for robotic manipulation.

8.5. Pose-based Flow Matching without IPA

ActionFlow [1] introduces two main changes compared to the previously presented methods: First, it increases the number of tokens by utilizing a single token per pose. This encodes information for the model and should be beneficial for manipulation tasks. On top of that, MHA is replaced with IPA. This step seems prudent, considering that the relative positioning between the objects and robot arms is crucial in manipulation tasks. The question is whether this is true or if this intrinsic structure limits the models' capabilities. A so-called *noIPA* model is implemented to examine this. It has the same structure as the ActionFlow model described in Section 8.4.4 and Figure 8.8, but the IPA block in the transformer encoder is replaced with a standard MHA block.

To make this work, the incoming tokens (\mathbf{p} , \mathbf{R} , \mathbf{f}) are processed as displayed in Figure 8.10: The rotation matrices \mathbf{R} are converted to quaternions \mathbf{q} before concatenating the positions \mathbf{p} . A MLP maps the 7-dimensional tokens to the embedding dimensional D_{emb} before combining them with the features \mathbf{f} and passing them to the MHA module.

With this implementation, the model still uses the more extensive set of tokens but can freely decide how to use the pose information. This comes at the cost of losing the invariance property. The effects of this are studied in Chapter 9 below.

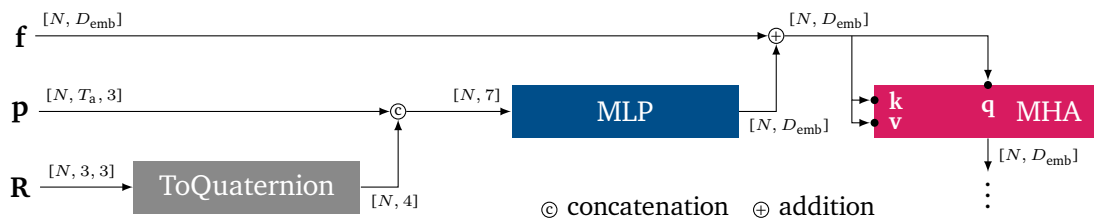


Figure 8.10.: The pose based attention block of the *noIPA* model, which replaces the IPA block in the ActionFlow model. After converting the rotations \mathbf{R} to quaternions \mathbf{q} , the concatenated pose information is embedded into the feature vector \mathbf{f} .

9. Experiments

Utilizing the developed framework for BIL, two key design questions, which are already examined by Funk *et al.* [1], are explored in more detail - for bimanual tasks:

- (a) Can representing each observed pose as an individual token enhance policy performance?
- (b) Does the IPA layer, which computes the relative poses between tokens, aid in uncovering informative features that improve policy performance, or is the IPA layer too restrictive?

Targeting an answer for (a), Diffusion Policy (DP) (Section 8.2) and FLOW Matching (Section 8.3) are compared against the ActionFlow (AF) (Section 8.4) and the *noIPA* model (Section 8.5) in various experiments. Question (b) will be answered by comparing ActionFlow and its modified version *noIPA*. In addition, the ActionFlow model’s invariance is tested to determine whether it can benefit limited demonstrations.

General Experiment Settings. All models are trained on datasets collected from waypoint-based expert demonstrations, where each task has a dataset comprising 1 000 successful demonstrations. Depending on the specific experiment, only a subset of this dataset might be utilized. Unless explicitly stated otherwise, experiments exclusively employ low-dimensional observations. Generally, these observations include end-effector position, orientation, gripper state, and object positions and orientations. Further details about the available low-dimensional observations are provided in Appendix A.

Hyperparameter settings for all models are comprehensively documented in Appendix B. Transformer-based components across the different experiments are configured to have a similar number of parameters. A cosine learning rate schedule with five warmup epochs is consistently employed. Depending on the task’s complexity, varying numbers of epochs are utilized. For diffusion models, DDIM uses 100 training steps combined with 25 inference steps, whereas the flow-based methods employ 100 training steps followed by

25 exponential inference steps. These relatively high values ensure that training and inference steps do not become limiting factors regarding model accuracy.

During training, the models are periodically saved and evaluated by executing several rollouts. Based on the performance in these preliminary evaluations, the best-performing model per seed is selected. A final, comprehensive evaluation is conducted by performing 100 rollouts for each selected model.

9.1. Sanity Check

To validate the quality of the waypoint-based datasets and gain a preliminary overview of each method’s performance, the four approaches are compared across the tasks *lift*, *simple-handover*, *place-ball*, *pick-place*, *hinged-bin*, *quad-insert*, and *transport*. The *peg-in-hole* task is excluded as it uses no grippers and, thus, does not depict a typical bimanual task. These experiments utilize 1 000 demonstrations per task during training. Due to the long horizon in the *transport*, only 500 are used. The models are periodically evaluated throughout training by conducting 10 rollouts at fixed intervals. Following training, the three best seeds and their corresponding best-performing checkpoints - selected based

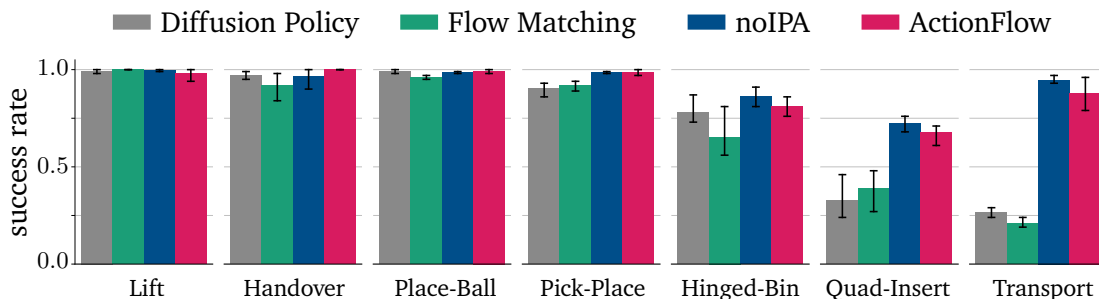


Figure 9.1.: Mean success rates for the models trained on 1 000 demonstrations each. Out of 5 seeds per model, the three best-performing model checkpoints run 100 rollouts, resulting in 300 rollouts per environment-model pair. To indicate the variance between the three models, the range of the individual success rates is marked with black. With increasing task complexity and length, the models with per-pose tokenization perform better. *noIPA* shows a minimal advantage over ActionFlow. The exact numerical values can be found in Table C.1.

on success rates - are evaluated comprehensively: Each chosen model undergoes a final evaluation of 100 rollouts, resulting in 300 rollouts per algorithm-task pair.

Figure 9.1 summarizes the results visually, while detailed numerical results can be found in Table C.1. Tasks with lower complexity, such as *lift*, *simple-handover*, and *place-ball*, exhibit very similar performances across all four approaches, with minor variations attributed primarily to statistical uncertainty. Similar to the results of [1], the Diffusion Policy and the Flow Matching model perform comparably when averaged across all tasks. The *pick-place* task, characterized by a longer horizon and additional subtasks - picking up a hammer, handover, and placing it into a target box - reveals some first limitations of the Diffusion Policy and Flow Matching compared to ActionFlow and *noIPA*. Tasks exhibiting high complexity, specifically *hinged-bin*, *quad-insert*, and *transport*, demonstrate clear advantages of ActionFlow and *noIPA*. This is especially evident in the *quad-insert* task with its stringent insertion tolerance of 1 mm and the *transport* task with its long horizon. Additionally, for these more complex tasks, *noIPA* shows a slight performance edge over ActionFlow.

As task complexity, accuracy demands, and task length increase, the benefits of per-pose tokenization become more pronounced.

9.2. Sample Efficiency

In this section, models are trained using a limited subset of the available demonstrations to investigate the sample efficiency. In addition to the models of Section 9.1, models are trained on 1, 4, 16, 64, and 256 successful demonstrations. The tasks *pick-place*, *hinged-bin*, and *quad-insert* are chosen to retrieve meaningful data: Each of the tasks requires the arms to interact differently. For the *pick-place* task, the arms have a physical coupling but move asynchronously, while the *hinged-bin* task has no direct interaction of the arms. Handling the lid and moving the hammer can be seen as individual movements with some temporal coupling. The *quad-insertion* has physical coupling and requires synchronized arm movement while grasping the bracket.

All models are trained for 1 500 epochs in case of the *pick-place* task and for 2 500 epochs on the *hinged-bin* and *quad-insert* task. After these training durations, the training loss and success rates stagnated. Figure 9.2 visualizes the results, and Table C.2 displays the results in detail. In the tasks with physical coupling (*pick-place* and *quad-insert*), the per-pose token models ActionFlow and *noIPA* outperform the models with fewer tokens, especially

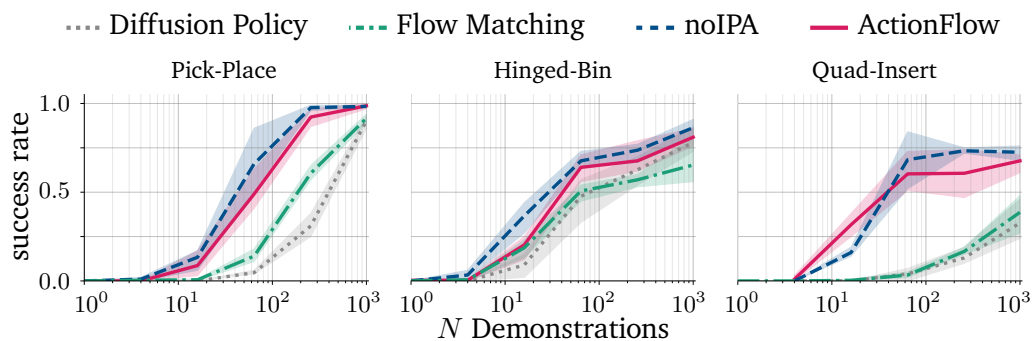


Figure 9.2.: Mean success rates for the models trained on 1, 4, 16, 64, 256 and 1000 demonstrations each. Each model is trained on three seeds and rolled out for 100 rollouts, resulting in 300 rollouts contributing to the means. The per-pose token models ActionFlow and *noIPA* show higher success rates for less available demonstrations. However, the difference to the less-token models is smaller in the *hinged-bin* task, which does not have physical coupling. Table C.2 holds the experimental results in detail.

in regimes with less available demonstrations. It can be seen that the difference in the *hinged-bin* task is more minor. This task does not contain physical coupling between the arms. The *quad-insertion* task requires symmetrical synchronization of the two arms. Here, ActionFlow performs slightly better than the *noIPA* model for $N = 16$ demonstrations. Yet, similar to Section 9.1, ActionFlow has a bit weaker performance than the *noIPA* model.

9.3. Invariance of ActionFlow

The key property and motivation of the ActionFlow architecture is the invariance to global transformations T_δ . To verify this, the following training-evaluation pipeline is used: The models are trained on demonstrations collected in the *lift* task using the waypoint expert but are evaluated on a rotated version. The transformation T_δ of the arms in the rotated setup contains a rotation by 90° and a translation to the table’s left edge. The same transformation is applied to the cups’ initial states. The *lift* task is easy to solve and chosen because its simplicity guarantees that a change in the success rate occurs from the rotation, not from general poor model performances. Figure 9.3 displays the training and evaluation setup with the corresponding experimental results. Detailed numerical results can be found in Table C.3. It should be clear that the rotated *lift* task is implemented only

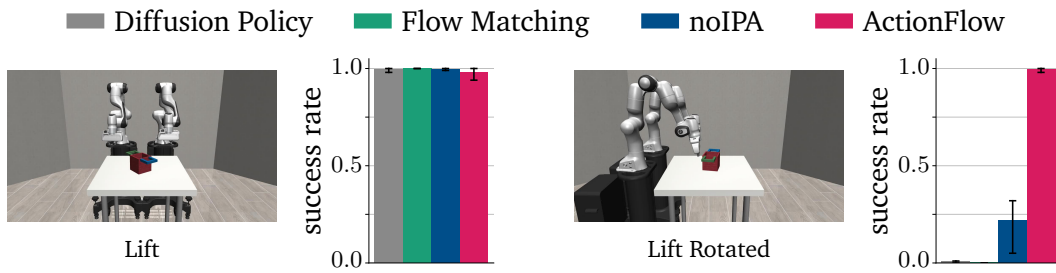


Figure 9.3.: Mean success rates for the models trained on *lift* demonstrations (left) and evaluated on a rotated version (right). Per model category, 300 rollouts (100 per model seed) contribute to the means. To indicate the variance between the three models, the range of the individual success rates is marked with black. The per-pose tokenization of *noIPA* enables some generalization, indicating that the learned features also consider the relative transformations between the individual poses. Still, only ActionFlow is invariant to global transformations. Refer to Table C.2 for detailed results.

to rotate the cup and the robots; the world frame is identical to the original version of the task, and all poses are expressed in this world frame.

The results are as expected: Only ActionFlow shows the invariance property. Still, it can also be seen that the *noIPA* model sometimes solves the task in the rotated setup. This indicates that this model also learned features considering the relative transformations between the individual poses. However, as the structure does not force this, the results are worse than ActionFlow's.

10. Discussion and Outlook

This thesis is structured into two key parts: Developing a BIL framework and its application. The first part of this work introduces a framework built upon two components of the ARISE initiative [77], namely ROBOSUITE [63] and ROBOMIMIC [54]. The second part applies this framework to evaluate various IL methods, with a particular focus on ActionFlow [1].

The presented BIL framework extends ROBOSUITE by incorporating four additional bimanual tasks: *place-ball*, *pick-place*, *quad-insert*, and *hinged-bin*. With ROBOSUITE’s backbone, the tasks are designed to be easily adaptable, allowing modifications such as changes in appearance or the use of different robot models. A waypoint-based expert is implemented to enable efficient data collection, facilitating the rapid generation of thousands of successful demonstrations. While this method ensures consistency, it lacks adaptability to variations and multi-modal behaviors. Future improvements could enhance this approach by incorporating multi-modal behaviors and further randomization to increase robustness. Additionally, while ROBOSUITE offers data collection methods primarily suited for single-arm tasks, extending support to bimanual teleoperation devices would be a valuable extension for future research.

Alongside the environment extensions, the second major component of the framework involves enhancements to ROBOMIMIC. This work integrates SOTA generative algorithms, specifically Diffusion Policy [9] and Flow Matching [34], into the framework. Additionally, ActionFlow [1] is implemented as a pose-based method, along with an alternative version (*noIPA*). The structured nature of ROBOMIMIC provides a flexible and effective environment for IL experiments, making it an excellent foundation for further research. However, the reliance on synthetic data collection within a simulated environment raises concerns about sim-to-real transferability, which could be explored further in future work. On top of that, future research could extend this framework by integrating additional IL and RL algorithms to expand its capabilities.

The second part of this work utilizes the developed framework to investigate ActionFlow in depth. Experiments are conducted specifically for bimanual tasks based on the original

work of Funk *et al.* [1]. ActionFlow leverages pose-based scene information and incorporates IPA to enforce invariance to global transformations. In addition to ActionFlow, the *noIPA* model is introduced as a second pose-based method. Unlike ActionFlow, *noIPA* processes pose information arbitrarily rather than using strictly relative transformations. The comparison of these methods with Diffusion Policy and Flow Matching, which are not specifically tailored for pose-based tasks, highlights a fundamental distinction: tokenization within the underlying Transformer architecture. While Diffusion Policy and Flow Matching assign a single token per sequence entry, ActionFlow and *noIPA* use a token per pose. With that, ActionFlow and *noIPA* enable explicit encoding of per-pose information, leading to richer scene representations.

Experiments presented in Chapter 9 answer two research questions: (a) Applying per-pose tokenization improves performance, as expected, by encoding additional structural information within the scene. It should be used if explicit pose information is available in the scene. (b) IPA enforces a more restrictive structure compared to MHA and the experimental results indicate that *noIPA* slightly outperforms ActionFlow. However, this difference is marginal. The invariance introduced by IPA might not provide an advantage in tasks where complete action trajectories are predicted, as consecutive predicted poses exhibit minimal relative transformations. Future research could explore whether the invariance property is beneficial in settings involving more significant relative transformations, such as policies predicting key points rather than whole trajectories.

Furthermore, the evaluation of the invariance property within ActionFlow was successful. This invariance property is particularly relevant for scenarios where end-effector poses cannot be directly derived from the robot’s internal kinematics, as with external gripping devices like UMI [95], with hand tracking for teleoperation [96], or when learning across different robotic platforms. In the first case, no robot base exists during data collection, making it hard to define a common frame for both end effectors. ActionFlow enables learning from these demonstrations without minding the absolute arrangement of the environment. In the second case, ActionFlow allows for faster and easier transfer between different robot platforms, as the transformations between the different robot bases do not have to be considered and estimated. These theoretical advantages exist, but further experiments are necessary to validate their benefits in practical applications.

The findings of this thesis have important implications for both research and practical applications in robotic IL. The developed framework provides a foundation for evaluating new learning algorithms in structured bimanual environments. This work offers a valuable tool for researchers working on robot learning methods. It enables flexible experiment rollouts and supports multiple IL approaches, which can be easily added. Additionally,

the insights from ActionFlow’s structure and invariance properties may influence future developments in tokenization strategies for pose-based policies.

Future research can build upon this work by further exploring invariance in key-point-based prediction, improving data collection methods, and assessing broader generalization capabilities across different robotic platforms. Ultimately, this work lays the foundation for continued advancements in BIL and structured action representations, enabling more robust and transferable policies.

References

- [1] N. Funk, J. Urain, J. Carvalho, V. Prasad, G. Chalvatzaki, and J. Peters, *ActionFlow: Equivariant, Accurate, and Efficient Policies with Spatially Symmetric Flow Matching*, 2024. arXiv: 2409.04576 [cs.LG] (cited on pp. i, ii, 2, 7, 9, 23, 25, 31, 35–39, 42, 45–47, 49, 52, 53).
- [2] J. Peters and S. Schaal, “Learning to Control in Operational Space”, *International Journal of Robotics Research (IJRR)*, pp. 197–212, 2008. DOI: 10.1177/0278364907087548 (cited on p. 4).
- [3] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation”, *IEEE Journal on Robotics and Automation*, pp. 43–53, 1987. DOI: 10.1109/JRA.1987.1087068 (cited on pp. 4, 13).
- [4] D. Oku and E. Obot, *Comparative Study Of PD, PI and PID Controllers For Control Of A Single Joint System In Robots*, Sep. 2019. DOI: 10.9790/1813-0709025154 (cited on pp. 4, 13).
- [5] T. Fujiki and K. Tahara, “Series admittance–impedance controller for more robust and stable extension of force control”, *ROBOMECH Journal*, Dec. 2022. DOI: 10.1186/s40648-022-00237-5 (cited on p. 4).
- [6] M. van Otterlo and M. Wiering, “Reinforcement Learning and Markov Decision Processes”, in *Reinforcement Learning: State-of-the-Art*. Springer Berlin Heidelberg, 2012, pp. 3–42, ISBN: 978-3-642-27645-3. DOI: 10.1007/978-3-642-27645-3_1 (cited on pp. 5, 26).
- [7] D. Pomerleau, “ALVINN: An Autonomous Land Vehicle In a Neural Network”, in *Proceedings of (NeurIPS) Neural Information Processing Systems*, 1989, pp. 305–313 (cited on pp. 6, 8, 23).
- [8] S. Kullback and R. A. Leibler, “On Information and Sufficiency”, *The Annals of Mathematical Statistics*, pp. 79–86, 1951. DOI: 10.1214/aoms/1177729694 (cited on p. 6).

-
-
- [9] C. Chi, Z. Xu, S. Feng, *et al.*, *Diffusion Policy: Visuomotor Policy Learning via Action Diffusion*, 2024. arXiv: 2303.04137 [cs.R0] (cited on pp. 7, 9, 23, 25, 27, 29, 30, 52, IX).
- [10] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, *Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware*, 2023. arXiv: 2304.13705 [cs.R0] (cited on pp. 7, 9, 11, 20).
- [11] M. Shridhar, L. Manuelli, and D. Fox, *Perceiver-Actor: A Multi-Task Transformer for Robotic Manipulation*, 2022. arXiv: 2209.05451 [cs.R0] (cited on pp. 7, 9, 18).
- [12] B. Argall, S. Chernova, M. Veloso, and B. Browning, “A Survey of Robot Learning from Demonstration”, *Robotics and Autonomous Systems*, pp. 469–483, 2009. DOI: 10.1016/j.robot.2008.10.024 (cited on p. 8).
- [13] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, “Survey of Imitation Learning for Robotic Manipulation”, *International Journal of Intelligent Robotics and Applications*, pp. 362–369, 2019. DOI: 10.1007/s41315-019-00103-5 (cited on p. 8).
- [14] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, “Imitation Learning: A Survey of Learning Methods”, *ACM Computing Surveys*, 2017. DOI: 10.1145/3054912 (cited on p. 8).
- [15] P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal, “Learning and generalization of motor skills by learning from demonstration”, in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 763–768. DOI: 10.1109/ROBOT.2009.5152385 (cited on p. 8).
- [16] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “Learning movement primitives”, in *Robotics Research. The Eleventh International Symposium*, P. Dario and R. Chatila, Eds., 2005, pp. 561–572. DOI: 10.1007/11008941_60 (cited on p. 8).
- [17] M. Drolet, S. Stepputtis, S. Kailas, *et al.*, *A Comparison of Imitation Learning Algorithms for Bimanual Manipulation*, 2024. arXiv: 2408.06536 [cs.R0] (cited on pp. 8, 11, 18, 20, V).
- [18] M. Grotz, M. Shridhar, T. Asfour, and D. Fox, *PerAct2: Benchmarking and Learning for Robotic Bimanual Manipulation Tasks*, 2024. arXiv: 2407.00278 [cs.R0] (cited on pp. 8, 9, 11, 16, 19, 20).
- [19] A. Dastider, H. Fang, and M. Lin, *APEX: Ambidextrous Dual-Arm Robotic Manipulation Using Collision-Free Generative Diffusion Models*, 2024. arXiv: 2404.02284 [cs.R0] (cited on pp. 8, 9, 11).
- [20] K. Shaw, Y. Li, J. Yang, *et al.*, *Bimanual Dexterity for Complex Tasks*, 2024. arXiv: 2411.13677 [cs.R0] (cited on pp. 8, 11).

-
-
- [21] J. Campbell and H. Ben Amor, “Bayesian Interaction Primitives: A SLAM Approach to Human-Robot Interaction”, in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, 2017 (cited on p. 8).
- [22] C. Bersch, B. Pitzer, and S. Kammel, “Bimanual robotic cloth manipulation for laundry folding”, in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 1413–1419. DOI: 10.1109/IRoS.2011.6048873 (cited on p. 8).
- [23] J. Grannen, Y. Wu, S. Belkhale, and D. Sadigh, *Learning Bimanual Scooping Policies for Food Acquisition*, 2022. arXiv: 2211.14652 [cs.R0] (cited on p. 8).
- [24] S. Mirrazavi, N. Figueroa, and A. Billard, “A unified framework for coordinated multi-arm motion planning”, *The International Journal of Robotics Research*, 2018. DOI: 10.1177/0278364918765952 (cited on p. 8).
- [25] C. Smith, Y. Karayiannidis, L. Nalpantidis, et al., “Dual Arm Manipulation — A Survey”, *Robotics and Autonomous Systems*, pp. 1340–1353, 2012. DOI: 10.1016/j.robot.2012.07.005 (cited on p. 8).
- [26] Y. Chen, T. Wu, S. Wang, et al., *Towards Human-Level Bimanual Dexterous Manipulation with Reinforcement Learning*, 2022. arXiv: 2206.08686 [cs.R0] (cited on p. 8).
- [27] S. Kataoka, S. K. S. Ghasemipour, D. Freeman, and I. Mordatch, *Bi-Manual Manipulation and Attachment via Sim-to-Real Reinforcement Learning*, 2022. arXiv: 2203.08277 [cs.R0] (cited on p. 8).
- [28] Y. Lin, A. Church, M. Yang, et al., *Bi-Touch: Bimanual Tactile Manipulation with Sim-to-Real Deep Reinforcement Learning*, 2023. arXiv: 2307.06423 [cs.R0] (cited on p. 8).
- [29] S. Stepputtis, M. Bandari, S. Schaal, and H. B. Amor, *A System for Imitation Learning of Contact-Rich Bimanual Manipulation Policies*, 2022. arXiv: 2208.00596 [cs.R0] (cited on pp. 8, 11).
- [30] S. Ross, G. J. Gordon, and J. A. Bagnell, *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning*, 2011. arXiv: 1011.0686 [cs.LG] (cited on p. 8).
- [31] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, *DART: Noise Injection for Robust Imitation Learning*, 2017. arXiv: 1703.09327 [cs.LG] (cited on p. 8).
- [32] A. Vaswani, N. Shazeer, N. Parmar, et al., *Attention Is All You Need*, 2023. arXiv: 1706.03762 [cs.CL] (cited on pp. 9, 23, 29, 31, 35, VII).

-
-
- [33] Z. Fu, T. Z. Zhao, and C. Finn, *Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation*, 2024. arXiv: 2401.02117 [cs.R0] (cited on pp. 9, 11).
- [34] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, *Flow Matching for Generative Modeling*, 2023. arXiv: 2210.02747 [cs.LG] (cited on pp. 9, 23, 25, 33–35, 52).
- [35] J. M. Jumper, R. Evans, A. Pritzel, *et al.*, “Highly accurate protein structure prediction with AlphaFold”, *Nature*, vol. 596, pp. 583–589, 2021. doi: 10.1038/s41586-021-03819-2 (cited on pp. 9, 41, 42).
- [36] J. Ho, A. Jain, and P. Abbeel, *Denoising Diffusion Probabilistic Models*, 2020. arXiv: 2006.11239 [cs.LG] (cited on pp. 9, 27, 34).
- [37] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox, *RVT: Robotic View Transformer for 3D Object Manipulation*, 2023. arXiv: 2306.14896 [cs.R0] (cited on p. 9).
- [38] J. Gao, X. Jin, F. Krebs, N. Jaquier, and T. Asfour, *Bi-KVIL: Keypoints-based Visual Imitation Learning of Bimanual Manipulation Tasks*, 2024. arXiv: 2403.03270 [cs.R0] (cited on pp. 9, 11).
- [39] J. Ho and S. Ermon, *Generative Adversarial Imitation Learning*, 2016. arXiv: 1606.03476 [cs.LG] (cited on p. 9).
- [40] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, *et al.*, *Generative Adversarial Networks*, 2014. arXiv: 1406.2661 [stat.ML] (cited on p. 9).
- [41] X. B. Peng, A. Kanazawa, S. Toyer, P. Abbeel, and S. Levine, *Variational Discriminator Bottleneck: Improving Imitation Learning, Inverse RL, and GANs by Constraining Information Flow*, 2020. arXiv: 1810.00821 [cs.LG] (cited on p. 9).
- [42] Y. Li, J. Song, and S. Ermon, “InfoGAIL: interpretable imitation learning from visual demonstrations”, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 3815–3825. doi: 10.5555/3294996.3295138 (cited on p. 9).
- [43] H. Xiao, M. Herman, J. Wagner, S. Ziesche, J. Etesami, and T. H. Linh, *Wasserstein Adversarial Imitation Learning*, 2019. arXiv: 1906.08113 [cs.LG] (cited on p. 9).
- [44] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, *Maximum entropy inverse reinforcement learning*, 2008. doi: 10.5555/1620270.1620297 (cited on p. 9).
- [45] B. Ziebart, J. Bagnell, and A. Dey, *Modeling Interaction via the Principle of Maximum Causal Entropy*, Jun. 2010 (cited on p. 10).

-
-
- [46] C. Finn, S. Levine, and P. Abbeel, *Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization*, 2016. arXiv: 1603.00448 [cs.LG] (cited on p. 10).
- [47] D. Garg, S. Chakraborty, C. Cundy, J. Song, M. Geist, and S. Ermon, *IQ-Learn: Inverse soft-Q Learning for Imitation*, 2022. arXiv: 2106.12142 [cs.LG] (cited on p. 10).
- [48] S. Reddy, A. D. Dragan, and S. Levine, *SQIL: Imitation Learning via Reinforcement Learning with Sparse Rewards*, 2019. arXiv: 1905.11108 [cs.LG] (cited on p. 10).
- [49] N. Gavenski, M. Luck, and O. Rodrigues, *Imitation Learning Datasets: A Toolkit For Creating Datasets, Training Agents and Benchmarking*, 2024. arXiv: 2403.00550 [cs.LG] (cited on p. 10).
- [50] A. Gleave, M. Taufeque, J. Rocamonde, *et al.*, *imitation: Clean Imitation Learning Implementations*, arXiv:2211.11972v1 [cs.LG], 2022. arXiv: 2211.11972 [cs.LG] (cited on p. 10).
- [51] R. Memmesheimer, I. Mykhalchyshyna, V. Seib, and D. Paulus, *Simitate: A Hybrid Imitation Learning Benchmark*, 2019. arXiv: 1905.06002 [cs.LG] (cited on p. 10).
- [52] S. Toyer, R. Shah, A. Critch, and S. Russell, *The MAGICAL Benchmark for Robust Imitation*, 2020. arXiv: 2011.00401 [cs.LG] (cited on p. 10).
- [53] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, *RLBench: The Robot Learning Benchmark & Learning Environment*, 2019. arXiv: 1909.12271 [cs.RO] (cited on pp. 10, 11).
- [54] A. Mandlekar, D. Xu, J. Wong, *et al.*, *What Matters in Learning from Offline Human Demonstrations for Robot Manipulation*, 2021. arXiv: 2108.03298 [cs.RO] (cited on pp. 10, 23–25, 52).
- [55] F. Al-Hafez, G. Zhao, J. Peters, and D. Tateo, *LocoMuJoCo: A Comprehensive Imitation Learning Benchmark for Locomotion*, 2023. arXiv: 2311.02496 [cs.LG] (cited on p. 10).
- [56] A. Mandlekar, S. Nasiriany, B. Wen, *et al.*, *MimicGen: A Data Generation System for Scalable Robot Learning using Human Demonstrations*, 2023. arXiv: 2310.17596 [cs.RO] (cited on pp. 10, 20).
- [57] X. Jia, D. Blessing, X. Jiang, *et al.*, *Towards Diverse Behaviors: A Benchmark for Imitation Learning with Human Demonstrations*, 2024. arXiv: 2402.14606 [cs.RO] (cited on pp. 10, 20).

-
-
- [58] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, *D4RL: Datasets for Deep Data-Driven Reinforcement Learning*, 2021. arXiv: 2004.07219 [cs.LG] (cited on p. 10).
- [59] A. Rajeswaran, V. Kumar, A. Gupta, *et al.*, *Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations*, 2018. arXiv: 1709.10087 [cs.LG] (cited on p. 10).
- [60] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, *Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning*, 2019. arXiv: 1910.11956 [cs.LG] (cited on p. 10).
- [61] J. Wong, A. Tung, A. Kurenkov, *et al.*, *Error-Aware Imitation Learning from Teleoperation Data for Mobile Manipulation*, 2021. arXiv: 2112.05251 [cs.RO] (cited on p. 10).
- [62] A. Mandlekar, Y. Zhu, A. Garg, *et al.*, *RoboTurk: A Crowdsourcing Platform for Robotic Skill Learning through Imitation*, 2018. arXiv: 1811.02790 [cs.RO] (cited on pp. 10, 20).
- [63] Y. Zhu, J. Wong, A. Mandlekar, *et al.*, *robosuite: A Modular Simulation Framework and Benchmark for Robot Learning*, 2022. arXiv: 2009.12293 [cs.RO] (cited on pp. 10, 12, 13, 16, 17, 20, 23, 24, 52).
- [64] E. Todorov, T. Erez, and Y. Tassa, “MuJoCo: A physics engine for model-based control”, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033. doi: 10.1109/IRoS.2012.6386109 (cited on pp. 10–12).
- [65] C. Sferrazza, D.-M. Huang, X. Lin, Y. Lee, and P. Abbeel, *HumanoidBench: Simulated Humanoid Benchmark for Whole-Body Locomotion and Manipulation*, 2024. arXiv: 2403.10506 [cs.RO] (cited on p. 10).
- [66] F. Xiang, Y. Qin, K. Mo, *et al.*, *SAPIEN: A Simulated Part-based Interactive Environment*, 2020. arXiv: 2003.08515 [cs.CV] (cited on p. 10).
- [67] J. Gu, F. Xiang, X. Li, *et al.*, *ManiSkill2: A Unified Benchmark for Generalizable Manipulation Skills*, 2023. arXiv: 2302.04659 [cs.RO] (cited on p. 10).
- [68] I. Garcia-Camacho, M. Lippi, M. C. Welle, *et al.*, “Benchmarking Bimanual Cloth Manipulation”, *IEEE Robotics and Automation Letters*, pp. 1111–1118, 2020. doi: 10.1109/LRA.2020.2965891 (cited on p. 11).
- [69] R. Ding, Y. Qin, J. Zhu, *et al.*, *Bunny-VisionPro: Real-Time Bimanual Dexterous Teleoperation for Imitation Learning*, 2024. arXiv: 2407.03162 [cs.RO] (cited on p. 11).

-
-
- [70] T. Zhang, D. Li, Y. Li, *et al.*, *Empowering Embodied Manipulation: A Bimanual-Mobile Robot Manipulation Dataset for Household Tasks*, 2024. arXiv: 2405.18860 [cs.R0] (cited on p. 11).
- [71] Y. Lin, A. Church, M. Yang, *et al.*, *Bi-Touch: Bimanual Tactile Manipulation with Sim-to-Real Deep Reinforcement Learning*, 2023. arXiv: 2307.06423 [cs.R0] (cited on p. 11).
- [72] E. Coumans and Y. Bai, *PyBullet, a Python module for physics simulation for games, robotics and machine learning*, URL: <http://pybullet.org>, 2021 (cited on p. 11).
- [73] F. Xie, A. Chowdhury, M. C. D. P. Kaluza, L. Zhao, L. L. S. Wong, and R. Yu, *Deep Imitation Learning for Bimanual Robotic Manipulation*, 2020. arXiv: 2010.05134 [cs.R0] (cited on p. 11).
- [74] E. Rohmer, S. P. N. Singh, and M. Freese, “V-REP: A versatile and scalable robot simulation framework”, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1321–1326. DOI: 10.1109/IROS.2013.6696520 (cited on p. 11).
- [75] K. Cetin, E. Tatlicioglu, and E. Zergeroglu, “On null-space control of kinematically redundant robot manipulators”, in *2016 European Control Conference (ECC)*, 2016, pp. 678–683. DOI: 10.1109/ECC.2016.7810367 (cited on p. 13).
- [76] H. Kasaei and M. Kasaei, *VITAL: Visual Teleoperation to Enhance Robot Learning through Human-in-the-Loop Corrections*, 2024. arXiv: 2407.21244 [cs.R0] (cited on p. 20).
- [77] *Advancing Robot Intelligence through Simulated Environments (ARISE)*, URL: <https://github.com/ARISE-Initiative>, 2025 (cited on pp. 23, 52).
- [78] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei, *Learning to Generalize Across Long-Horizon Tasks from Human Demonstrations*, 2021. arXiv: 2003.06085 [cs.R0] (cited on p. 23).
- [79] S. Fujimoto and S. S. Gu, *A Minimalist Approach to Offline Reinforcement Learning*, 2021. arXiv: 2106.06860 [cs.LG] (cited on p. 23).
- [80] I. Kostrikov, A. Nair, and S. Levine, *Offline Reinforcement Learning with Implicit Q-Learning*, 2021. arXiv: 2110.06169 [cs.LG] (cited on p. 23).
- [81] A. Mandlekar, F. Ramos, B. Boots, *et al.*, *IRIS: Implicit Reinforcement without Interaction at Scale for Learning Control from Offline Robot Manipulation Data*, 2020. arXiv: 1911.05321 [cs.R0] (cited on p. 23).

-
-
- [82] D. Reynolds, “Gaussian Mixture Models”, in *Encyclopedia of Biometrics*, S. Z. Li and A. Jain, Eds. Boston, MA: Springer US, 2009, pp. 659–663, ISBN: 978-0-387-73003-5. DOI: 10.1007/978-0-387-73003-5_196 (cited on p. 23).
- [83] D. P. Kingma and M. Welling, “An Introduction to Variational Autoencoders”, *Foundations and Trends® in Machine Learning*, vol. 12, pp. 307–392, 2019, ISSN: 1935-8245. DOI: 10.1561/22000000056 (cited on p. 23).
- [84] R. M. Schmidt, *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*, 2019. arXiv: 1912.05911 [cs.LG] (cited on p. 23).
- [85] Y. Song and S. Ermon, *Generative Modeling by Estimating Gradients of the Data Distribution*, 2020. arXiv: 1907.05600 [cs.LG] (cited on p. 27).
- [86] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*, 2015. arXiv: 1503.03585 [cs.LG] (cited on p. 27).
- [87] J. Song, C. Meng, and S. Ermon, *Denosing Diffusion Implicit Models*, 2022. arXiv: 2010.02502 [cs.LG] (cited on pp. 27, 29, 34).
- [88] C. Luo, *Understanding Diffusion Models: A Unified Perspective*, 2022. arXiv: 2208.11970 [cs.LG] (cited on p. 28).
- [89] T. Fjelde, E. Mathieu, and V. Dutordoir, *An Introduction to Flow Matching*, URL: <https://mlg.eng.cam.ac.uk/blog/2024/01/20/flow-matching.html>, 2024 (cited on p. 33).
- [90] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, *Neural Ordinary Differential Equations*, 2019. arXiv: 1806.07366 [cs.LG] (cited on p. 33).
- [91] J. Yim, A. Campbell, A. Y. K. Foong, *et al.*, *Fast protein backbone generation with SE(3) flow matching*, 2023. arXiv: 2310.05297 [q-bio.QM] (cited on pp. 37, 38).
- [92] J. Yim, B. L. Trippe, V. D. Bortoli, *et al.*, *SE(3) diffusion model with application to protein backbone generation*, 2023. arXiv: 2302.02277 [cs.LG] (cited on p. 37).
- [93] J. Solà, J. Deray, and D. Atchuthan, *A micro Lie theory for state estimation in robotics*, 2021. arXiv: 1812.01537 [cs.RO] (cited on pp. 37, 38).
- [94] B. C. Hall, “The Baker-Campbell-Hausdorff Formula”, in *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Springer New York, 2003, pp. 63–90, ISBN: 978-0-387-21554-9. DOI: 10.1007/978-0-387-21554-9_3 (cited on p. 38).

-
-
- [95] C. Chi, Z. Xu, C. Pan, *et al.*, *Universal Manipulation Interface: In-The-Wild Robot Teaching Without In-The-Wild Robots*, 2024. arXiv: 2402.10329 [cs.RO] (cited on p. 53).
- [96] F. Zhang, V. Bazarevsky, A. Vakunov, *et al.*, *MediaPipe Hands: On-device Real-time Hand Tracking*, 2020. arXiv: 2006.10214 [cs.CV] (cited on p. 53).
- [97] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization*, 2019. arXiv: 1711.05101 [cs.LG] (cited on p. IX).

A. Tasks in Detail

Giving information about the task implementations, the following sections display the details of the bimanual tasks available in the BIL framework (Chapter 5). The task collection includes the ROBOSUITE tasks *peg-in-hole*, *lift handover*, and *transport* as well as the additional implemented tasks *place-ball*, *pick-place*, *quad-insert* and *hinged-bin*. A general overview categorizing the tasks by coupling and required arm synchronization can be found in Table 5.1.

A.1. Peg-In-Hole

Task Description. In this task, the arms' grippers are replaced with a fixed cylindrical stick and a squared plate with a squared hole. The goal is to insert the stick into the hole.

Success Criteria. The task is completed once the stick is pushed through the squared hole by at least half its length.

Coupling and Synchronization. The coupling is mainly spatial; both arms work within a common workspace. If the task is solved as intended, no physical coupling appears. Whether temporal coupling is present depends on how the insertion is fulfilled, but it is unnecessary. The required synchronization between the two arms is not symmetric but must be synchronized to place the two parts relative to each other.

Task Randomization. During the initialization of the task, the robot's joint configuration is slightly randomized by applying Gaussian noise $\mathcal{N}(0, 0.02)$ to each joint.

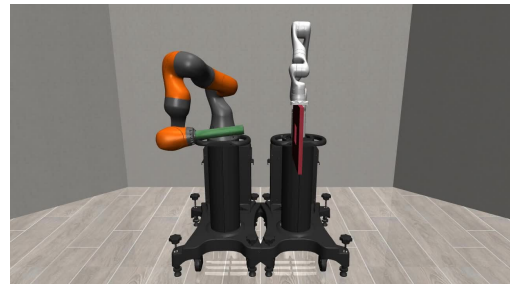


Figure A.1.: Peg-In-Hole task.

Low Dimensional Observations. The available low-dimensional observations are hole position, hole quaternion, hole position, and peg quaternion. These values are given in the world frame.

A.2. Lift

Task Description. The task requires the robot arms to pick up a box-shaped cup from the table. Two edge handles allow for an easy grasp of the cup.

Success Criteria. The task is completed if the cup is lifted by 10 cm.

Coupling and Synchronization. The coupling in this task is spatial and physical: The two arms share the same workspace and are coupled once both grippers grasp the cup. From this moment, symmetric synchronization is required, too.

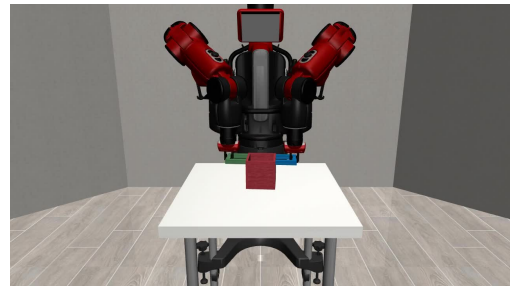


Figure A.2.: Lift task.

Task Randomization. The robot's joint configuration is slightly randomized by applying Gaussian noise $\mathcal{N}(0, 0.02)$ to each joint. The cup is positioned randomly in a 6×6 cm large square between the two arms. The orientation of the cup is varied by $\pm 60^\circ$.

Low Dimensional Observations. The available low-dimensional observations are cup position, cup quaternion, and the positions of the two handles. All are given in the world frame.

A.3. Handover

Task Description. One arm must pick up a hammer and hand it to the other. With the *handover-simple* task, a variant is available where the hammer is already placed in the robot's right gripper.

Success Criteria. To complete the task, the receiving arm must grasp the hammer handle at least 10 cm above the table, and the other arm cannot touch the hammer.

Coupling and Synchronization. Besides the spatial and physical coupling, this task also has temporal coupling, as the handover movement follows a sequential sequence. The required synchronization between the arms is asymmetric but synchronized.

Task Randomization. Each initial joint position is randomized by $\mathcal{N}(0, 0.02)$. The hammer is placed in front of the right arm in a 20×10 cm region and can be oriented in any orientation (360°). Additionally, the hammer can be flipped, meaning that the tip can point clockwise or counter-clockwise. The handle thickness is varied between 3 and 4 cm, and the handle length is sampled to be between 10 and 25 cm.

Low Dimensional Observations. The available low-dimensional observations (given in the world frame) are the hammers' position and quaternion.

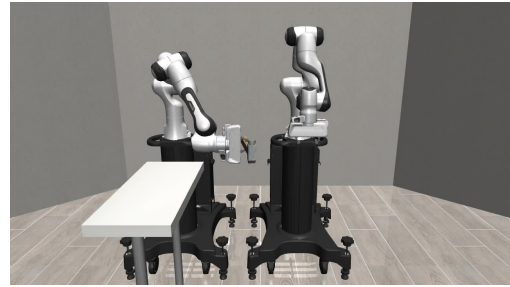


Figure A.3.: Handover task.

A.4. Transport

Task Description. This task has the longest horizon, consisting of multiple sub-tasks. The overall exercise is to place the hammer in the rear box. Before that, the right robot arm must remove the lid covering the box that contains the hammer at the start of the episode, and the left arm must remove the red dice from the rear box to the box up front. As the right arm cannot reach the target box, a handover of the hammer must be performed.

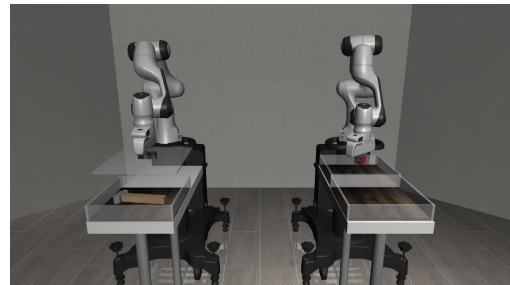


Figure A.4.: Transport task.

Success Criteria. The task is completed if the red cub is placed in the box up front and the hammer is placed in the rear box.

Coupling and Synchronization. The task contains all (spatial, physical, temporal) couplings and requires synchronized coordination.

Task Randomization. The robot’s joint configuration is slightly randomized by $\mathcal{N}(0, 0.02)$. The positions of all objects and boxes are sampled from 1×1 cm regions. The hammer’s initial rotation is varied by $\pm 54^\circ$. Its size is fixed to a thickness of 3 cm and a handle length of 20 cm. The orientation of the red cube is varied by $\pm 30^\circ$.

Low Dimensional Observations. The available low-dimensional observations are given in the world frame: Cube position and quaternion, hammer position and quaternion, position and quaternion of the lids handle, positions of the target bin and the bin up front.

A.5. Place-Ball

Task Description. The robot arms must lift and place a large ball in a box in front of the ball. The ball is designed so large that grippers cannot grasp it.

Success Criteria. If the ball is inside the box and the arms are not touching the ball, the task is completed.

Coupling and Synchronization. The coupling in this task is spatial and requires careful symmetric coordination to prevent dropping the ball. As with all bimanual tasks, synchronization between the two arms is required.

Task Randomization. The robot’s initial joint positions are randomized by $\mathcal{N}(0, 0.02)$. The ball’s and box’s initial positions are sampled from 10×10 cm regions. Additionally, the orientation of the box is varied by $\pm 180^\circ$.

Low Dimensional Observations. Ball position, box position, and box quaternion are the available low-dimensional observations. The values are given in the world frame.

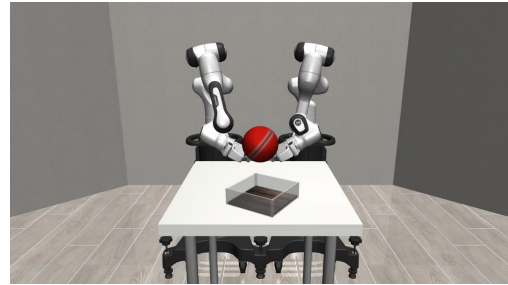


Figure A.5.: Place-Ball task.

A.6. Pick-Place

Task Description. This task requires a hammer to be placed in a target box. The box and the hammer are placed so that a handover of the hammer is required before it can be placed in the box.

Success Criteria. The task is accomplished once the hammer is placed in the box and no arm is grasping it.

Coupling and Synchronization. The coupling and coordination are equivalent to the *handover* task (spatial, physical, temporal and asymmetric, synchronized).

Task Randomization. The robot's joint configuration is slightly randomized by applying Gaussian noise $\mathcal{N}(0, 0.02)$ to each joint. The hammer's and box's initial positions are sampled from 10×10 cm regions. The orientations are varied by $\pm 180^\circ$, and the hammer's thickness and length are varied between 3 to 4 and 22 to 25 cm. The hammer is always in front of the right arm, and the box is always in front of the left arm.

Low Dimensional Observations. Given in the world frame, the available low dimensional observations are hammer position/orientation and box position/orientation.



Figure A.6.: Pick-Place task.

A.7. Quad-Insert

Task Description. This task is adapted from Drolet *et al.* [17]. A bracket with four holes and two handles must be placed on four pins. The insertion tolerance for each pin is only 1 mm.

Success Criteria. The task is accomplished once the bracket is placed on the stand.

Coupling and Synchronization. The coupling is spatial and physical. The required coordination is synchronized and highly symmetric as soon as the bracket is lifted.

Task Randomization. Noise $\mathcal{N}(0, 0.02)$ is used to randomize the robot’s initial joint configuration slightly. The bracket and stand positions are sampled from 4×4 cm large regions. The rotations are varied by $\pm 15^\circ$. The bracket is always positioned in front of the right arm, and the stand is always in front of the left arm.

Low Dimensional Observations. The available low-dimensional observations (world frame) are bracket position/quaternion, positions of the handles, and stand position/quaternion.



Figure A.7.: Quad-Insert task.

A.8. Hinged-Bin

Task Description. The arms must work together to place the hammer in a bin covered with a hinged lid. The opening angle of the lid is limited such that the box does not stay open on its own.

Success Criteria. To complete the task, the hammer must be placed in the box, and its lid must be closed.

Coupling and Synchronization. The task exhibits temporal and spatial coupling. Required coordination is asymmetric and synchronized, with each arm having distinct but interdependent roles.

Task Randomization. The robot’s joint configuration is slightly randomized by applying Gaussian noise $\mathcal{N}(0, 0.02)$. The hammer’s and bin’s initial positions are sampled from 10×10 cm regions. The orientations are randomized by $\pm 22.5^\circ$. The hammer thickness and length are varied between 3 to 4 and 10 to 15 cm. The hammer is always in front of the right arm, and the bin is always in front of the right arm, with its general orientation so that the open side points to the center of the table.

Low Dimensional Observations. Available are hammer position/quaternion, bin position/quaternion, and handle position/quaternion. All low-dimensional values are given in the world frame.



Figure A.8.: Hinged-Bin task.

B. Hyperparameters

The following sections describe the chosen hyperparameters for the models and experiments of Chapter 9. It is explicitly stated that no fine-tuning has been carried out. The values were selected so that all algorithms work and are kept as equal as possible. More precise analysis could increase performance and efficiency, but this work did not focus on these issues.

B.1. Models

All presented algorithms are implemented as multi-step policies and use a Transformer architecture [32] as the underlying model.

Horizons. All models take in a sequence of the last $T_o = 4$ observations to predict the next $T_p = 8$ actions. Of these predictions, only the first ($T_a = 1$) action is executed before a new prediction sequence is generated. The procedure is visualized in Figure 8.1.

Training and Inference. Both underlying mechanisms (diffusion and flow) use fewer inference steps during sampling than training. The Diffusion Policy uses 100 diffusion timesteps during training but runs DDIM inference with 25 timesteps. The flow-based methods use a linear schedule with 100 steps during training but an exponential schedule with 35 steps during inference. These parameters are not optimized but chosen so that they do not limit the policy performances. Reducing the time steps used might make faster inference times possible without sacrificing performance.

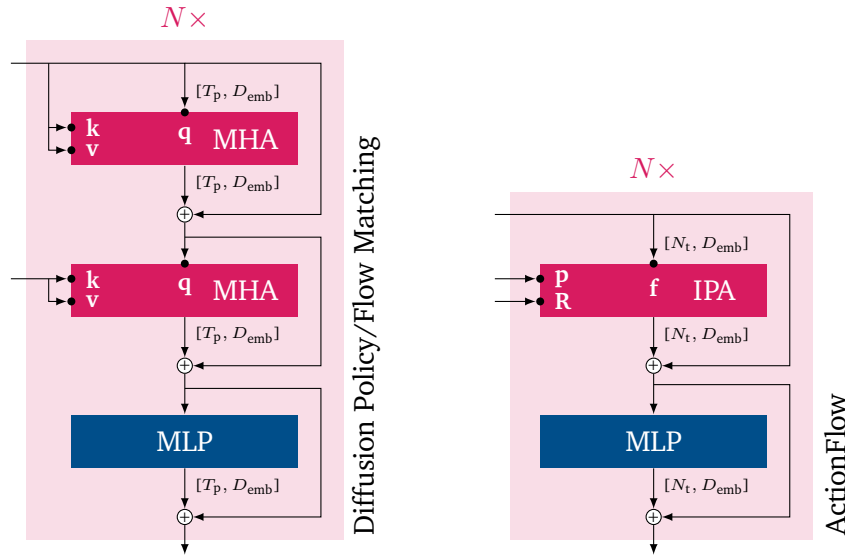


Figure B.1.: Comparison of the Transformer blocks used in the Diffusion Policy/Flow Matching (left) and ActionFlow (right). The block for the *noIPA* model is equivalent to the one of ActionFlow, but IPA is replaced with MHA - see Figure 8.10.

Transformer Structures. The underlying Transformer models of the algorithms consist of multiple blocks with MHA/IPA and feedforward, see Figures 8.3 and 8.8. Figure B.1 extracts and visualizes the relevant components. Key design choices are the number of blocks N , the embedding dimensionality D_{emb} , and the number of heads N_{head} . The values are chosen equally for all models with

$$N = 4, D_{\text{emb}} = 64, \text{ and } N_{\text{head}} = 4.$$

Tuning these values to the limit could again increase training and inference time.

B.2. Training

The algorithms are trained to minimize the mean squared error between the target noise removal/velocity and the predicted values - see Equations (8.2) and (8.6). For ActionFlow and *noIPA*, the rotation, translation, and gripper components of the loss are weighted with $w_p = 2$, $w_r = 1$, and $w_{\text{grip}} = 2$, see Equation (8.14).

	Lift	Handover	Place-Ball	Pick-Place	Hinged-Bin	Quad-Insert	Transport
# Demos	1 000	1 000	1 000	1 000	1 000	1 000	1 000
# Steps	78 467	102 787	182 060	221 570	317 030	227 868	226 645
# Epochs (9.1)	400	600	600	1 000	2 500	2 000	3 000
# Epochs (9.2)	—	—	—	1 500	2 500	2 500	—

Table B.1.: *Top:* Overview of all datasets generated with the waypoint experts. The total number of demonstrations and the total number of action steps in the datasets are provided. *Bottom:* Number of epochs used for training the experiments 9.1 and 9.2.

Durations. Table B.1 displays how many epochs are used to train the models in the experiments 9.1 and 9.2. In experiment 9.1, all available demonstrations (collected with the waypoint experts) are used. Depending on the length of the task, different numbers of total update steps per epoch are applied. The sample efficiency experiment (Section 9.2) uses fewer demonstrations. Experiment 9.3 uses the trained *lift* models from experiment 9.1 and uses them in the rotated setup.

Optimizers. The training utilizes the AdamW optimizer [97]. For the flow-based methods $\beta_1 = 0.9$, $\beta_2 = 0.95$ and a L2 regularization of 10^{-4} are used. The Diffusion Policy uses a L2 regularization of 10^{-3} for the noise prediction components and 10^{-6} for the observation encoder [9]. Mini-batch updates of size 128 are used. The learning rate follows a cosine learning rate schedule with five warmup epochs. From 10^{-4} , the learning rate decays continuously till the final epoch. Figure B.2 visualizes the learning rate schedule.

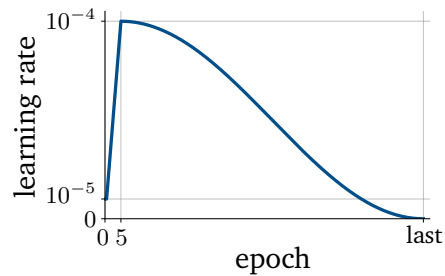


Figure B.2.: Cosine learning rate schedule with five warmup epochs, an initial learning rate of 10^{-4} , and a warmup decay factor of 0.1.

C. Experimental Results in Detail

Remark. Success rates are reported as the mean of three models trained on different seeds. The *mean* value of the three models is provided. In this case, a standard deviation would not be meaningful. Instead, the maximum and minimum success rates are always specified in the format $mean_{min}^{max}$. This information indicates the variance. Diffusion Policy, Flow Matching, and Action Flow are abbreviated with DP, FM, and AF.

C.1. Sanity Check

Table C.1 provides the success rates for Section 9.1. The corresponding visualization can be found in Figure 9.1. Models are trained to solve the tasks *lift*, *handover*, *ball-insert*, *pick-place*, *hinged-bin*, *quad-insert*, and *transport*.

	Lift	Handover	Place-Ball	Pick-Place	Hinged-Bin	Quad-Insert	Transport
DP	0.99 $\frac{1.00}{0.98}$	0.97 $\frac{0.99}{0.95}$	0.99 $\frac{1.00}{0.98}$	0.90 $\frac{0.93}{0.86}$	0.78 $\frac{0.87}{0.73}$	0.33 $\frac{0.46}{0.24}$	0.27 $\frac{0.29}{0.24}$
FM	1.00 $\frac{1.00}{1.00}$	0.92 $\frac{0.98}{0.84}$	0.96 $\frac{0.97}{0.95}$	0.92 $\frac{0.94}{0.89}$	0.65 $\frac{0.81}{0.56}$	0.39 $\frac{0.48}{0.27}$	0.21 $\frac{0.24}{0.19}$
noIPA	1.00 $\frac{1.00}{0.99}$	0.97 $\frac{1.00}{0.90}$	0.98 $\frac{0.99}{0.98}$	0.98 $\frac{0.99}{0.98}$	0.86 $\frac{0.91}{0.81}$	0.72 $\frac{0.76}{0.68}$	0.95 $\frac{0.97}{0.93}$
AF	0.98 $\frac{1.00}{0.94}$	1.00 $\frac{1.00}{1.00}$	0.99 $\frac{1.00}{0.98}$	0.99 $\frac{1.00}{0.97}$	0.81 $\frac{0.86}{0.76}$	0.68 $\frac{0.71}{0.61}$	0.88 $\frac{0.96}{0.79}$

Table C.1.: Detailed results of the sanity check experiment in Section 9.1, which are visualized in Figure 9.1. Out of 5 seeds per model, the three best-performing model checkpoints run 100 rollouts, resulting in 300 rollouts per mean. To indicate the variance between the three models, the range of the individual success rates is donated as $mean_{min}^{max}$.

C.2. Sample Efficiency

To test the sample efficiency of the three model types, they are trained on $N \in \{1, 4, 16, 64, 256, 1\,000\}$ demonstrations. To provide a good cross-section of various appearing couplings, the tasks *pick-place*, *hinged-bin*, and *quad-insert* are selected. Table C.2 displays the results, which are visualized in Figure 9.2.

N Demonstrations		1	4	16	64	256	1 000
Pick-Place	DP	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.01}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.05 $\frac{0.06}{0.04}$	0.31 $\frac{0.38}{0.25}$	0.90 $\frac{0.93}{0.86}$
	FM	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.01}{0.00}$	0.01 $\frac{0.01}{0.00}$	0.14 $\frac{0.18}{0.11}$	0.61 $\frac{0.65}{0.57}$	0.92 $\frac{0.94}{0.89}$
	noIPA	0.00 $\frac{0.00}{0.00}$	0.01 $\frac{0.02}{0.00}$	0.13 $\frac{0.17}{0.06}$	0.66 $\frac{0.86}{0.51}$	0.98 $\frac{0.99}{0.96}$	0.98 $\frac{0.99}{0.98}$
	AF	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.09 $\frac{0.17}{0.03}$	0.49 $\frac{0.63}{0.41}$	0.92 $\frac{0.97}{0.87}$	0.99 $\frac{1.00}{0.97}$
Hinged-Bin	DP	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.09 $\frac{0.15}{0.02}$	0.48 $\frac{0.57}{0.33}$	0.63 $\frac{0.70}{0.54}$	0.78 $\frac{0.87}{0.73}$
	FM	0.00 $\frac{0.00}{0.00}$	0.01 $\frac{0.01}{0.00}$	0.19 $\frac{0.23}{0.13}$	0.51 $\frac{0.54}{0.45}$	0.57 $\frac{0.60}{0.53}$	0.65 $\frac{0.81}{0.56}$
	noIPA	0.00 $\frac{0.00}{0.00}$	0.03 $\frac{0.06}{0.01}$	0.37 $\frac{0.44}{0.27}$	0.68 $\frac{0.73}{0.64}$	0.74 $\frac{0.77}{0.67}$	0.86 $\frac{0.91}{0.81}$
	AF	0.00 $\frac{0.01}{0.00}$	0.00 $\frac{0.01}{0.00}$	0.20 $\frac{0.25}{0.12}$	0.64 $\frac{0.71}{0.56}$	0.68 $\frac{0.79}{0.62}$	0.81 $\frac{0.86}{0.76}$
Quad-Insert	DP	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.04 $\frac{0.07}{0.00}$	0.13 $\frac{0.19}{0.10}$	0.33 $\frac{0.46}{0.24}$
	FM	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.01}{0.00}$	0.03 $\frac{0.05}{0.02}$	0.17 $\frac{0.19}{0.15}$	0.39 $\frac{0.48}{0.27}$
	noIPA	0.00 $\frac{0.00}{0.00}$	0.00 $\frac{0.00}{0.00}$	0.16 $\frac{0.19}{0.14}$	0.68 $\frac{0.84}{0.52}$	0.73 $\frac{0.75}{0.72}$	0.72 $\frac{0.76}{0.68}$
	AF	0.00 $\frac{0.00}{0.00}$	0.01 $\frac{0.03}{0.00}$	0.32 $\frac{0.38}{0.25}$	0.60 $\frac{0.73}{0.51}$	0.61 $\frac{0.74}{0.47}$	0.68 $\frac{0.71}{0.61}$

Table C.2.: Detailed results of the sample efficiency experiment in Section 9.2, which are visualized in Figure 9.2. The table displays the mean success rates for the models trained with different numbers of demonstrations N . Trained on three different seeds, each model runs 100 rollouts, resulting in 300 rollouts per mean value. To indicate the variance between the three models, the range of the individual success rates is donated as $mean \frac{max}{min}$.

C.3. Invariance

The following experiment is set up to verify the invariance property of ActionFlow and to check the other models regarding this property: The models are trained on the *lift* task. Afterward, they are evaluated on two setups. The first one is equivalent to the one during training. In the second one, the robot arms are placed on the left side of the table. This is equivalent to rotating the arms by 90° and moving them to the table’s edge via a translation. The same transformation is applied to the cups’ initial states. Table C.3 provides the numerical values displayed in Figure 9.3.

	Lift		Rotated Lift	
DP	0.99	$\frac{1.00}{0.98}$	0.01	$\frac{0.01}{0.00}$
FM	1.00	$\frac{1.00}{1.00}$	0.00	$\frac{0.00}{0.00}$
noIPA	1.00	$\frac{1.00}{0.99}$	0.22	$\frac{0.32}{0.05}$
AF	0.98	$\frac{1.00}{0.94}$	0.99	$\frac{1.00}{0.98}$

Table C.3.: Detailed results of the invariance experiment in Section 9.3, which are visualized in Figure 9.3. The table displays the mean success rates for the models trained in the *lift* task. The models are evaluated on a setup identical to training and on a setup that rotated the cup and the robots by 90 degrees. Trained on three different seeds, each model runs 100 rollouts, resulting in 300 rollouts per mean value. To indicate the variance between the three models, the range of the individual success rates is denoted as $mean \begin{smallmatrix} max \\ min \end{smallmatrix}$.