ELIE ALJALBOUT

LEARNING STATE AND ACTION SPACES FOR ROBOT LEARNING

Technische Universität München
TUM School of Computation, Information and Technology

LEARNING STATE AND ACTION SPACES FOR ROBOT LEARNING

ELIE ALJALBOUT

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr.-Ing. Klaus Diepold
Prüfende der Dissertation:	1.	Prof. Dr. Rudolph Triebel
	2.	Prof. Dr.-Ing. Matthias Althoff
	3.	Prof. Dr. Davide Scaramuzza

Die Dissertation wurde am 18.12.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 05.07.2024 angenommen.

ABSTRACT

Robot learning is a powerful paradigm for skill acquisition in robotics. Learning such skills based on data can be very expensive in terms of interactions with the environment, especially when learning via reinforcement. This is mainly due to the complexity of processing robotic observations and the challenges in identifying dynamics and controlling the system. These challenges rendered robot learning inefficient and impeded its practical application in the real world.

This thesis addresses this problem from the perspectives of perception and control. In the context of reinforcement learning, the primary emphasis is on the choice of state and action spaces. The state space defines the information on which a control policy bases its decision. Since robotic tasks often involve complex and high-dimensional observations, it is crucial to reduce these into a compact state representation that is beneficial for control. The action space defines the level of abstraction at which the policy commands the robot. Robot actuators typically expect smooth control commands adhering to certain embodiment constraints. An ideal action space for robot learning should strike the right balance between abstracting difficult aspects of the action generation process and allowing the policy to have full control over the robot's behavior.

The first part of this thesis proposes methods for devising state spaces suitable for robot manipulation learning. These methods involve either designing or learning the state representations. Designing state representations constitutes a choice of information deemed necessary for the tasks at hand, such as objects' positions and methods for obtaining this information from the available observations. For learning state representations, we use methods from self-supervised machine learning to identify different factors of variations in robot interaction data.

In the second part of the thesis, we propose methods for building and learning action representations for robot manipulation learning. We first study the role of the action space in learning manipulation policies and transferring them from simulation to the real world. We build action spaces based on inductive biases, with the aim of abstracting general concepts from the policy based on well-established principles from robot motion planning and control. Furthermore, we further learn action representations using self-supervised machine learning methods, thereby extending the applicability of robot learning to even more challenging multi-robot tasks.

ZUSAMMENFASSUNG

Roboterlernen ist ein leistungsfähiges Paradigma für den Erwerb von Fähigkeiten in der Robotik. Das Erlernen von Fähigkeiten auf der Grundlage von Daten kann in Bezug auf die Interaktion mit der Umgebung sehr kostspielig sein, insbesondere wenn das Lernen über Verstärkung erfolgt. Dies ist vor allem auf die Komplexität der Verarbeitung von Roboterbeobachtungen und die Herausforderungen bei der Identifizierung der Dynamik und der Steuerung des Systems zurückzuführen. Diese Herausforderungen machen das Roboterlernen ineffizient und erschweren die praktische Anwendung in der realen Welt.

In dieser Arbeit wird diese Problemstellung aus der Perspektive der Perzeption und Steuerung behandelt mit dem Schwerpunkt auf der Wahl der Zustands- und Aktionsräume. Der Zustandsraum definiert die Informationen, auf deren Grundlage eine Policy ihre Entscheidung trifft. Da Roboteraufgaben oft komplexe und hochdimensionale Beobachtungen beinhalten, ist es entscheidend, diese in eine kompakte Zustandsrepräsentation zu reduzieren, die für die Steuerung von Vorteil ist. Der Aktionsraum definiert die Abstraktionsebene, auf der die Policy den Roboter steuert. Roboteraktuatoren erwarten in der Regel glatte Steuerbefehle, die bestimmte Beschränkungen während der Ausführung einhalten. Ein idealer Aktionsraum für das Lernen von Robotern sollte das richtige Gleichgewicht zwischen der Abstraktion schwieriger Aspekte des Aktionsgenerierungsprozesses und der vollen Kontrolle der Policy über das Verhalten des Roboters herstellen.

Im ersten Teil dieser Arbeit werden Methoden zur Ableitung von Zustandsräumen vorgeschlagen, die für das Lernen von Robotermanipulation geeignet sind. Diese Methoden beinhalten entweder den manuellen Entwurf oder das Lernen von Zustandsrepräsentationen. Der Entwurf von Zustandsrepräsentationen besteht aus einer Auswahl von Informationen, die für die jeweilige Aufgabe als notwendig erachtet werden. Des Weiteren beinhalten sie Methoden, um diese Informationen aus den verfügbaren Beobachtungen zu erhalten. Zum Erlernen der Zustandsrepräsentation verwenden wir Methoden des Selbstüberwachten maschinellen Lernens, um verschiedene Faktoren in den Roboterinteraktionsdaten zu identifizieren.

Im zweiten Teil der Arbeit werden Methoden für den Aufbau und das Lernen von Handlungsrepräsentationen für das Manipulationslernen von Robotern vorgeschlagen. Zunächst wird die Rolle des Aktionsraums beim Erlernen von Manipulationsaufgaben und deren Übertragung von der Simulation in die reale Welt untersucht. Die Aktionsräume werden auf der Grundlage induktiver Verzerrungen auf-

gebaut, mit dem Ziel, allgemeine Konzepte aus der Policy zu abstrahieren, die auf gut etablierten Prinzipien der Bewegungsplanung und -steuerung von Robotern basieren. Darüber hinaus erlernen wir Aktionsräume mit Methoden des Selbstüberwachten maschinellen Lernens, wodurch die Anwendbarkeit des Roboterlernens auf noch anspruchsvollere Multi-Roboter-Aufgaben erweitert wird.

PUBLICATIONS

This thesis is based on ideas, experiments, and material that appeared in the following publications and (under-review) papers:

Elie Aljalbout, Ji Chen, Konstantin Ritt, Maximilian Ulmer, and Sami Haddadin. "Learning Vision-based Reactive Policies for Obstacle Avoidance." In: *Proceedings of the 2020 Conference on Robot Learning (CoRL)*. Vol. 155. Proceedings of Machine Learning Research. PMLR, 2021.

Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. "On the role of the action space in robot manipulation learning and sim-to-real transfer." In: *IEEE Robotics and Automation Letters* (2024).

Elie Aljalbout, Maximilian Karl, and Patrick van der Smagt. "CLAS: Coordinating Multi-Robot Manipulation with Central Latent Action Spaces." In: *Proceedings of The 5th Annual Learning for Dynamics and Control Conference (L4DC)*. 2023.

Elie Aljalbout, Maximilian Ulmer, and Rudolph Triebel. "Seeking Visual Discomfort: Curiosity-driven Representations for Reinforcement Learning." In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 3591–3597.

Axel Sauer*, Elie Aljalbout*, and Sami Haddadin. "Tracking Holistic Object Representations." In: *British Machine Vision Conference (BMVC)*. 2019.

The publication at the British Machine Vision Conference received the *best science paper award*. Additionally, the following is a list of papers and publications that were created during this work, but are not incorporated in this manuscript:

Elie Aljalbout. "Dual-Arm Adversarial Robot Learning." In: *Conference on Robot Learning (CoRL)*. PMLR. 2022, pp. 1814–1819.

Marvin Alles and Elie Aljalbout. "Learning to centralize dual-arm assembly." In: *Frontiers in Robotics and AI* 9 (2022), p. 830007.

Nirnai Rao*, Elie Aljalbout*, Axel Sauer*, and Sami Haddadin. "How to make deep RL work in practice." In: *Deep RL Workshop, Conference on Neural Information Processing Systems (NeurIPS)* (2020).

Maximilian Ulmer, Elie Aljalbout, Sascha Schwarz, and Sami Haddadin. "Learning robotic manipulation skills using an adaptive force-impedance action space." In: *arXiv preprint arXiv:2110.09904* (2021).

* indicates shared first authorship.

ACKNOWLEDGMENTS

First, I would like to thank my supervisors Rudolph and Patrick. Thank you both for the great opportunities you enabled for me, for being open yet critical, and for being great mentors throughout this journey.

I would also like to thank my colleagues and collaborators at Volkwagen MLRL, in particular Ahmed, Alex, Botond, Felix, Justin, Marvin, Max K., Nutan, Nikos, Philip, and Xingyuan. I am extremely grateful for your proactive support, the great discussions, joint work, and table tennis games we have had together. I am glad to have been part of such a unique environment.

Special thanks are due to Sami, as well as to my colleagues, students, and collaborators at MIRMI, in particular Ali, Axel, Carlos, Christian, Diego, Erfan, Felipe, Ioannis, Ji, Konstantin, Lennart, Luis, Max U., Nirnai, and Riddhiman. My career in robotics research started with this group of people, and being part of this environment helped shape my perspective on the field.

Finally, my biggest thanks goes to my family: my parents, sisters, and Karen. Your support and understanding during this period made it so much easier and so much more exciting.

CONTENTS

ACRONYMS

AE	autoencoder
CL	contrastive learning
DDPG	deep deterministic policy gradient
DoF	degrees of freedom
DMP	dynamic movement primitives
ELBO	evidence lower bound
KL	Kullback-Leibler divergence
LSTM	long short-term memory network
ML	machine learning
MDP	Markov decision process
MARL	multi-agent reinforcement learning
PG	policy gradient
PI	policy iteration
PPO	proximal policy optimization
RAE	regularized autoencoder
RL	reinforcement learning
RMP	Riemannian motion policies
SAC	soft actor-critic
SRL	state representation learning
TRPO	trust region policy optimization
TD_3	twin delayed deep deterministic policy gradient
VI	value iteration
VAE	variational autoencoders

Part I

BACKGROUND

1 INTRODUCTION

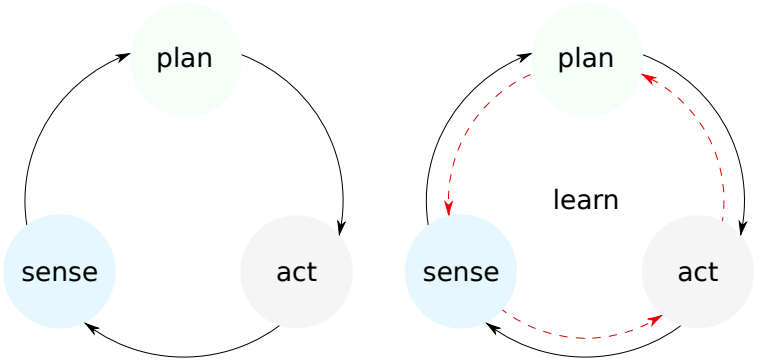


Figure 1.1: Moving away from the sense-plan-act paradigm towards a more learning-centric sense-plan-act-learn approach.

1.1 FROM ROBOTICS TO ROBOT LEARNING

One of the main challenges in robotics is to bring advanced robots outside of structured environments, such as research labs and factories, into more human-like environments, ultimately integrating them into our daily lives. For this to happen, robots should be capable of properly understanding their surroundings, planning, reasoning about their actions, and making optimal decisions. These different capabilities form the sense-plan-act cycle. While this paradigm can be considered outdated, most of today's architectures include these same components.

This architecture is implemented using modular pipelines with parts, separately, handling different aspects of the problem. Modules usually include sensor data acquisition, perception, task/motion planning, and low-level control. However, one major drawback of this modularity is that small errors occurring in one module could significantly disadvantage the functioning of others. Without a way to learn from these errors, they are bound to happen again during the robot's lifetime. These errors are in many cases the result of a bad system design, unrealistic assumptions, mis-tuned parameters, or simply an edge case that was never accounted for. All of which can be traced down to engineering faults.

For many years now, researchers have been developing smart algorithms with the goal of reducing these errors to a tolerable and practical threshold. All of these advances made robots safer and more

capable, and enabled new applications. Yet, with some minor exceptions, most robots deployed in the real world still function in well-structured environments and have a limited set of skills.

Simultaneously, artificial intelligence has recently seen tremendous progress, mainly in machine learning (ML). ML algorithms have led to state-of-the-art performance in computer vision [49, 58, 103] and natural language processing [46, 129, 153, 180, 194] benchmarks but still struggle to perform well on some very simple robotic applications. However, learning offers the means of eliminating the errors resulting from engineering faults and also reduces the effect of these errors since different modules could benefit from each other's learning objectives. Additionally, learning reduces the cost of programming robotic skills, and hence the need of a human in the loop. In contrast to the traditional approach of engineering every component needed for a certain new skill, a robot could learn it based on a simple specification such as a reward or cost function. This has led to greater interest and efforts in robot learning as a field that leverages ML methods to tackle complex robotics problems. This paradigm shift is illustrated in Figure 1.1.

1.2 CHALLENGES IN ROBOT LEARNING

Far from being a solved problem, robot learning presents multiple challenges. For starters, there are multiple ways to learn control policies for robots. For instance, imitation learning relies on trajectories of human demonstrations to obtain a policy. These trajectories vary in size and type depending on the method. Due to the passive nature of these methods, their performance is highly dependent on the provided dataset i.e., agents learned with imitation learning have no way of actively seeking information beyond the support of the dataset. On the other hand, an agent could learn a control policy via direct interaction with its environment using reinforcement learning (RL). RL methods do not require pre-recorded datasets of action-observation pairs, but instead learn to associate such pairs by maximizing the cumulative reward of the environment. This process requires in practice a well-defined reward or cost function for the environment and the task at hand. This aspect makes RL less accessible than imitation learning methods since defining a proper reward function arguably requires more engineering knowledge than providing demonstrations. This problem can be alleviated by a mere combination of these paradigms, where the reward function is learned from demonstrations and the policy is then learned via interaction using RL. This paradigm is referred to as inverse RL [1, 135]. Nonetheless, robot learning still suffers to find its way into real-world applications. This is due to multiple challenges, some of which I'll discuss next.

Sample Efficiency. Common to all these paradigms is the problem of sample inefficiency, i.e., requiring a large amount of data to

obtain good policies. This problem can be tolerated in virtual (simulated) environments. However, it can be quite expensive in real-world robotics scenarios, such as the ones relevant to this thesis. This inefficiency is due to multiple factors. The most popular one is related to the large state and action spaces encountered in such environments. Modern-day robots have high degrees of freedom (DoF) and are capable of high-frequency actuation. In addition, the states of such environments are deduced based on measurements from different sensors, such as joint encoders, force-torque sensors, and cameras. Learning to process such observations is itself a very inefficient problem. On top of all that, observations and actions are usually noisy and continuous, making the training of such policy harder due to the resulting variance in the training gradients. Furthermore, robotics environments have very complex dynamics due to physical components, such as friction and contact. Capturing this complexity with function approximators requires a lot of samples.

Exploration. Large amounts of interaction samples can still lead to bad policies if they mostly contain redundant and irrelevant information. Quantity and quality of data are both required in robot learning [33]. Hence, another challenge is exploration. An agent's actions are considered exploratory, if its pure purpose is to seek novelty and not to solve the task at hand. Designing mechanisms to choose such actions is itself a hard problem, especially when dealing with large state and action spaces. Another challenge is to integrate such exploratory mechanisms in the policy learning process. When only taking exploratory actions, learning a policy becomes harder since useful actions become very rare. This problem constitutes the exploration-exploitation dilemma.

Generalization & Robustness. Related to the last two challenges are generalization and robustness. Trained with finite data, the policy could struggle to generalize beyond the support of the dataset. It would also struggle to deal with unseen disturbances. Exposing the policy to large and diverse state-action pairs is the naive yet not guaranteed way, since there are no metrics yet to assess whether a certain amount of data is enough or contains diverse enough samples to capture the dynamics of a task. Instead, one could aim to embed inductive biases into the policy structure and training method. The challenge in this case is to define these inductive biases in a way that does not affect the applicability of robot learning algorithms to a wide variety of tasks and applications.

Autonomous Randomized Experiments. Furthermore, beyond being expensive, collecting data in the real world could be challenging. First, the robot should be able to try out random actions in its environment without harming itself or its surroundings. This aspect usually requires well-engineered safety features to be embedded in the environment controllers, reward functions, and reset mechanisms. These design decisions should also account for task feasibility. To illustrate

that, let us assume a robot is learning an object-pushing skill. During an arbitrary episode, the object could be kicked outside of the robots' workspace, and hence made unreachable. Resetting such an environment would require human intervention. Alternatively, physical constraints could be built into the environment, as to prevent such a thing from happening. Such mechanisms could reduce the need for a human in the loop, but come with a large engineering cost.

Interpretability. Another challenge is interpretability. Most learning-based approaches rely on function approximators such as neural networks for policy representation. These models offer little transparency as why and how a given input was mapped to a certain output, which is why they are usually criticized as being black-box models. This problem becomes more critical for decision-making problems, since the outputs, in this case, correspond to actions that will affect the environment. It is then desirable to be able to reason about these actions and their generation process. Imagine a robot manipulator collaborating with a human on a handover task. Without proper care, the robot could end up harming the human. In such scenarios, an understanding of the action generation mechanism becomes necessary.

Reproducability. Recently, it has been shown that experimental results in RL are hard to reproduce. Simple hyperparameters or implementation differences could lead to significant discrepancies in performance by the same algorithm [72, 155]. In practice, these findings could hint that deploying RL algorithms is itself a different problem than tuning the methods offline or in a test environment. Additionally, many RL algorithms tend to lead to high variance in performance even under the same configuration. This makes deployment even trickier. All of these aspects motivate the development of novel robot learning methods that are reliable in real-world scenarios.

1.3 THESIS CONTRIBUTIONS

With these challenges in mind, the goal of this thesis is to devise robot learning approaches that are suitable and applicable for real-world robotic manipulation. Before going into the contributions of this thesis, we will illustrate the details of the studied problem with an example. Let's consider a robot arm that is learning to push a cube to a target position. To achieve this goal, the robot should have access to its own state as well as the state of the cube. The cube' state would ideally contain positional information about the cube, potentially with higher-order-derivative variables such as velocity and acceleration. In a real-world scenario, this information can be obtained based on remote sensing, e.g., vision. Hence, a robot learning to perform this task would first need to learn to understand and process its visual sense. Equipped with this skill, the robot then needs to act on its environment, in a way that serves the task at hand. Modern robot

manipulators are typically torque-controlled. Hence, to actuate the robot, the agent would need to determine the right torques to apply to the robot joints based on the previously inferred state. Such a mapping would at least require some implicit notion of the inverse kinematics and dynamics of the robot. In the classical RL sense, an agent would need to learn these perception and action problems based on a very weak reward signal. To alleviate this problem, this thesis looks into inductive biases that can be embedded into different components of robot manipulation learning. For instance, for perception, this thesis presents methods for supervised and self-supervised training of visual processing modules. These methods learn a state space representation based on visual observations. These representations can either be learned jointly or separately from the policy training. The main goal of these methods is to reduce the sample requirements of robot learning, and improve exploration. Similarly, the thesis presents approaches to design and learn action space representations, that simplify the control aspect of learning a task. These methods simplify the action space in which the policies need to act, making it either less complex, more abstract, or at least more directly suitable for certain families of tasks. In addition, this thesis presents methods to leverage and extend these concepts to multi-agent settings as a way to scale robot learning to multi-robot learning. Finally, this work includes simulated and real-world evaluation of these methods.

1.4 THESIS STRUCTURE

The next part of this thesis introduces the theoretical foundations upon which the main thesis work is built. Chapter 2 formally introduces reinforcement learning and some of the RL methods used in this thesis. Part ii and Part iii contain the main contributions of this thesis. Part ii discusses state representations for robotics. Chapter 3 introduces some background on state representation learning, and the theory and notation behind methods used in the following chapters. Chapter 4 presents a supervised learning method for visual tracking. The approach augments recent advances in object tracking with an elegant mathematical memory selection mechanism for gathering pseudo-ground truth templates at inference time. Chapter 5 proposes an approach for perception-driven exploration in RL. The proposed method is simple yet effective. It encourages agents to take actions that lead to observations which its own perception module still struggles to process. Part iii discusses action representations for robotics. Chapter 6 introduces popular action spaces in the literature, and presents a study on the performance of different action space choices on exploration, policy capability, and sim-to-real transfer. Chapter 7 proposes a novel motion-centric action space for robotic manipulation. The proposed space embeds concepts from robot motion generation and differential geometry to abstract general features of motion that

the policy does not necessarily need to handle in most manipulation tasks. Chapter 8 introduces an approach for learning centralized latent action spaces for decentralized control. These central latent action spaces are trained to contain information related to the task and not to the individual robots, which is beneficial for scaling multi-robot manipulation control. Finally, the last part of the thesis contains a discussion on the topic of robot learning, a conclusion of the thesis, open challenges, and potential future research directions.

2

PRELIMINARIES

In recent years, RL has seen many successes in multiple domains such as games [131, 171], chip design [130], and robotics [13, 81]. In this thesis, reinforcement learning is the primary method used to learn control policies for robot manipulation. Hence, to ensure the self-contained comprehensiveness of this thesis, we will introduce the basics of reinforcement learning and the theory behind the algorithms used throughout the different chapters. Most of the material is based on textbooks [25, 181] and online resources [2]. Throughout the text, there are references to the original publications. Readers who are familiar with state-of-the-art RL methods can safely move on to the next chapter. Nevertheless, a quick look at the notation remains beneficial for all types of readers.

2.1 INTRODUCTION TO REINFORCEMENT LEARNING

RL methods constitute a subfamily of machine learning approaches in which an agent learns to make sequential decisions by engaging with its environment. The goal is to maximize a cumulative reward signal through a trial-and-error process. At each time step, the agent receives an observation or state of the environment and takes a control action according to this data. The agent then receives feedback in the form of a reward or penalty. Over time, the agent adjusts its behavior in a way that maximizes its total reward. Repeating this behavior enables the agent to reach effective decision-making strategies for the task it is optimizing.

2.1.1 *Formalism*

The most central part of each RL agent is its **policy**. In fact, the terms agent and policy are often used interchangeably in the RL literature. The policy is tasked with outputting a control action given a state or observation of the environment. We typically use the symbol π to refer to an RL policy. There are multiple characteristics of the policies used in this paradigm.

For example, a policy could either be deterministic or stochastic. In the deterministic case, the policy outputs an action given a state $\mathbf{s}_t$ or observation $\mathbf{o}_t$: $\mathbf{a}_t = \pi(\mathbf{s}_t)$. In contrast, a stochastic policy outputs a distribution of actions from which the actual control action can be sampled: $\mathbf{a}_t \sim \pi(.|\mathbf{s}_t)$. Here, a common choice is the Gaussian distribution. Thus, the policy could output a mean and a variance to fully parameterize the distribution.

Furthermore, a policy could either be parameterized or non-parameterized. The latter typically means maintaining a look-up table that contains the optimal actions for each possible state of the system. A parameterized policy is a parameterized function that receives as input the state or observation of the environment and outputs a suitable action. In deep RL, this function is a deep neural network. However, there are other options. In robotics, motion primitives are common. Such primitives usually embed some useful inductive bias in the representation of the policy. For example, dynamic movement primitives (DMP) are inspired by dynamical systems and model the policy using attractor dynamics and a forcing function [82, 162]. Other approaches, such as probabilistic movement primitives [145] and Riemannian motion policies [157] use concepts from probability theory and differential geometry, respectively. We use the following notation for a parameterized policy π_ϕ, where ϕ refers to the parameters that define this policy according to the chosen representation. In the case of a neural network, ϕ corresponds to the weights of the network.

Furthermore, a policy can either be recurrent or not. The recurrence itself could be with respect to either the state or action. The Markov property assumed in the RL context can relax the requirements for recurrent policies. Nevertheless, recurrence can put more emphasis on certain information from the previous states or help enforce properties such as smoothness in the cases of state and action recurrence, respectively. Of course, this comes at the cost of making the training less stable [21] and introducing additional hyperparameters.

Each RL task comes with a distinct **state space** $\mathcal{S}$. The state of the environment consists of all variables needed to describe it. In the RL case, we define the state to contain all information needed to control the system. However, in real-world scenarios, the state of an environment is not directly accessible. Instead, we could observe the state of the system through various sensory measurements. If we have direct access to the state or it is possible to fully infer the state from the observations, we refer to the system as fully observable. This condition is, in practice, not easy to meet. Real-world sensors are limited in terms of resolution, perspective, precision, and what they can directly measure. For instance, high-order derivatives of variables such as acceleration or jerk are difficult to measure directly with a sensor. Such environments are called partially observable. In robotics, we typically (partially) observe the state through various sensors that measure data about the robot itself (via proprioceptive sensors) and the environment (via exteroceptive sensors). Both states and observations can be either discrete or continuous depending on the environment and sensors used.

In addition to the state space, each task defines its own **action space** $\mathcal{A}$. The action is nothing but a control command that the system expects at each time step. Similarly to states, actions can also be deterministic or continuous. This depends on the actuation mechanism

of the environment. For example, video games typically expect discrete commands that describe some movement or atomic actions. In physical systems such as robots, most native control commands typically have continuous values. For example, the lowest level of control available for most commercial robot arms is either a vector of torques or electric currents that are distributed to the different joints of the arm. A policy that acts in such a low-level action space would need to (at least intuitively) understand the dynamics of the system. Alternatively, the action space can be simplified by embedding manually designed controllers as part of the environment. If we again consider robot manipulation, we could make the actions be desired joint positions or velocities, and use a well-established robot control algorithm to output torque that would lead to these desired variables. This simplifies and even limits the role of the policy, which in the latter case would only need to understand the kinematics of the robot and some high-level intuitive physical properties. This comes at the cost of the policy losing control over the low-level behavior.

Another central part of RL is the **reward function**. The reward is a function of the state and action. It outputs a value that describes how good it is to be at that state and the cost incurred by taking that action. In RL, the reward is the main way to describe a desired task or behavior. It can be sparse or dense. An example of a sparse reward is one that only incentivizes distinct states that enable the task such as the success state. A dense reward typically describes the objective in a more continuous way. For example, for a goal-reaching task, a dense reward could be defined as the negative distance to the goal.

Given the state, action, and reward, we can now define the Markov decision process (MDP). It is the main formalism to mathematically describe control systems in the RL literature. A finite-horizon, discounted MDP is characterized by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$. The state and action spaces are denoted respectively by $\mathcal{S}$ and $\mathcal{A}$. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denotes the transition dynamics and $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward. In addition to these terms, we have an initial state distribution ρ_0, a discount factor $\gamma \in [0, 1]$, and a horizon of length T. The optimal policy $\pi : \mathcal{S} \to \mathcal{A}$, maximizes the expected discounted reward

$$J(\pi) = \mathbb{E}_{\tau \sim p_\tau}\left[\sum_{t=0}^{T-1} \gamma^t r(s_t, a_t)\right] \tag{2.1}$$

$$= \mathbb{E}_{\tau \sim p_\tau}[R(\tau)], \tag{2.2}$$

where τ refers to trajectories of the form $(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots \mathbf{s}_{T-1}, \mathbf{a}_{T-1})$, sampled from a trajectory distribution p_τ, and $R(\tau)$ is cumulative trajectory reward, also called the return.

As the name suggests, MDPs adhere to the Markov property. The Markov property states that the future state of a system or process is only determined by its current state, and the action taken at that point

in time, but is not affected by the sequence of events that preceded it. This means that the system has no recollection of past states apart from its current one. This property can be characterized by

$$p(\mathbf{s}_{t+1} \mid \mathbf{s}_{0:t}, \mathbf{a}_{0:t}) = p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t). \tag{2.3}$$

While the reward function describes how good it is to be in a given state in the short-term perspective, it is of interest to know the value of being in that state in the long-term perspective. For this reason, RL algorithms define a **value function**. This function defines the expected cumulative reward when starting at a given state and following some policy,

$$V^{\pi}(\mathbf{s}_t) = \mathbb{E}_{\tau \sim p_\tau}\left[\sum_{t=i}^{T-1} r(\mathbf{s}_t, \mathbf{a}_t)\right]. \tag{2.4}$$

The optimal policy π^* satisfies

$$\forall \mathbf{s} \in \mathcal{S}; \quad V^{\pi^*}(\mathbf{s}) = \max_{\pi} V^{\pi}(\mathbf{s}). \tag{2.5}$$

We can now introduce the Bellman equation,

$$V^{\pi}(\mathbf{s}_t) = \mathbb{E}_{\tau \sim p_\tau}[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^{\pi}(\mathbf{s}_{t+1})]. \tag{2.6}$$

When written in this form, we can clearly see the recurrent nature of the algorithm induced by this equation. This equation is the backbone of RL algorithms.

For certain RL algorithms — namely, model-based RL — it is essential to maintain a **sequential model** describing the dynamics of the environment. These models can be given in rare cases. However, in most cases, if needed, such a model needs to be learned using data collected from the system. Once this model is available, it can be used to plan optimal actions or help to learn the optimal policy. The latter is either done by using the model to sample additional states, actions, and rewards to train the policy, or by backpropagating the policy gradients through this model.

2.1.2 *Paradigms and Methods*

With these components in mind, we can now classify RL methods into families of methods. First, we can differentiate between value and policy-based methods. Value-based methods focus on deriving the value function. The optimal policy is then simply the one that selects the actions that lead to the highest value. Due to this selection process, value-based methods are only feasible for environments with discrete state and action spaces. On the contrary, policy-based methods directly learn the optimal policy itself, without explicitly estimating state values. These methods are applicable to environments with discrete and continuous state and action spaces. Another major

theoretical advantage of policy-based methods is that they directly optimize the RL objective from equation (2.1). A sub-category of policy-based methods learns a policy with the help of a learned value function. These methods are called actor-critic methods. In this context, the actor refers to the policy, and the critic is the value function. Since the focus of this thesis is on robotics, it only uses policy-based methods as well as actor-critic since they can handle the continuous action spaces usually encountered in such environments. Another important distinction is based on the access to a sequential dynamics model. Model-free approaches do not assume access to such a model, while model-based methods either assume the model is available or learn it using data. In this thesis, we exclusively used model-free methods. The main reason for this is that the dynamics of robot manipulation tasks are very complex and include a lot of discontinuities, especially at contact. This makes learning a good sequential model quite challenging and learning a downstream policy even more challenging with such a low-quality model. We can further distinguish RL methods based on multiple other features (exact vs. approximate methods, on-policy vs off-policy algorithms). However, this section only discusses the two distinctions that were actively considered when choosing the methods used in this thesis. A comprehensive overview and taxonomy of RL methods can be found in [2, 181].

2.2 VALUE-BASED METHODS

The fundamental principle that guides these methods is the Bellman equation introduced in equation (2.6). With this equation, we can derive the value iteration (VI) algorithm that inspired most value-based RL algorithms. We first assume discrete state and action spaces. As its name suggests, VI involves an iterative process. Specifically, given an arbitrary policy π we iteratively update the value function V^π as follows. We start with an arbitrary initial value function V_0^π. We then iterate a process, where at each iteration i, we update the value function,

$$V_{i+1}^\pi(\mathbf{s}_t) = \sum_{\mathbf{a}_t \in \mathcal{A}} \pi(\mathbf{a}_t \mid \mathbf{s}_t) \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V_i^\pi(\mathbf{s}_{t+1})]. \tag{2.7}$$

This process is repeated until convergence. Similarly, we can find the optimal value function $V^* = V^{\pi^*}$ by instead iterative updating the optimal value function,

$$V_{i+1}^*(\mathbf{s}_t) = \max_{\mathbf{a}_t \in \mathcal{A}} \left(\sum_{\mathbf{s}_{t+1} \in \mathcal{S}} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V_i^*(\mathbf{s}_{t+1})] \right). \tag{2.8}$$

Note that this process does not require any access to the optimal policy π^*. This process can be updated for continuous state and action

spaces, by using a parametrized value function V_δ and updating the parameters δ of the value function at each iteration using the error between the newly computed value estimates V^*_{i+1} and the old ones V^*_i,

$$\delta_{i+1} = \delta_i + \alpha \nabla \delta_i (V^*_{i+1}(\mathbf{s}_t, \delta_i) - V^*_i(\mathbf{s}_t, \delta_i)), \tag{2.9}$$

where α is a learning rate, and $V^*_{i+1}(\mathbf{s}_t, \delta_i)$ is computed as shown in equation (2.8), while using $V^*_i(\mathbf{s}_t, \delta_i)$.

2.2.1 *Q-Learning*

While VI can converge to the (optimal) value function [25, 181], it does not directly yield the optimal policy. Given the optimal value function V^*, acting optimally would correspond to picking actions according to the following objective,

$$\underset{\mathbf{a}_t \in \mathcal{A}}{\operatorname{argmax}} \big(\mathbb{E}_{\mathbf{s}_{t+1} \sim p(.|\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V^*(\mathbf{s}_{t+1})] \big). \tag{2.10}$$

However, in practice, this is only feasible if we have access to the environment dynamics $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, and if the state and action space dimensions are sufficiently small for this objective to be tractable.

However, in Q-learning, a similar process is proposed, using a value called the action value. The action value $Q(\mathbf{s}, \mathbf{a})$ defines the expected value of being at state $\mathbf{s}$ and taking action $\mathbf{a}$. Mathematically, it can be formulated as follows,

$$Q(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1} \sim p(.|\mathbf{s}_t, \mathbf{a}_t)} [r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V(\mathbf{s}_{t+1})]. \tag{2.11}$$

Given this definition, Q-learning introduces an iterative process similar to VI, to obtain the action value function or table using interactions with the environment. Similarly to VI, to learn the action value, we start with an arbitrary initial action value Q_0. At each step, we sample an action according to the current estimate Q_i. This typically corresponds to using an ϵ-greedy policy, which is a policy that selects random actions with probability ϵ, or otherwise selects optimal actions based on Q_i. With the resulting $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ data, we can update the estimate of Q based on the following objective,

$$\begin{aligned} Q_{i+1}(\mathbf{s}_t, \mathbf{a}_t) = Q_i(\mathbf{s}_t, \mathbf{a}_t) + \alpha [r(\mathbf{s}_t, \mathbf{a}_t) \\ + \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} Q_i(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - Q_i(\mathbf{s}_t, \mathbf{a}_t)]. \end{aligned} \tag{2.12}$$

Intuitively, this update moves the Q values slightly in the direction of a VI-style estimate that benefits from the new information gained about the reward $r(\mathbf{s}_t, \mathbf{a}_t)$. This process is repeated until convergence, or for a fixed number of iterations. After convergence, acting optimally would correspond to choosing actions as

$$\underset{\mathbf{a} \in \mathcal{A}}{\operatorname{argmax}} Q(\mathbf{s}, \mathbf{a}). \tag{2.13}$$

Unlike VI, this algorithm does not require access to the dynamics of the environment $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. Nevertheless, it is only tractable for environments that have discrete state and action spaces.

2.2.2 *Deep Q-Learning*

Many real-world environments have continuous or high-dimensional state spaces. For instance, any robot manipulation task requires position-like information in the state. To handle such requirements, deep Q-learning uses function approximators, such as deep neural networks, to approximate the action-value function [131]. The action-value function is then defined as a neural network, called the deep Q-network, or in short-form DQN. We can now write the action-value function with its parameters as $Q_\omega(s, a)$. We can then update the parameters ω of the network using a mean-squared-error loss between a target Q-value and the current estimate. The target Q-value, referred to as Y, is computed based on the Bellman equation,

$$Y = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a}_{t+1} \in \mathcal{A}} Q_\omega(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}). \tag{2.14}$$

As before, we can now interact with the environment to collect data. This interaction can again be based on an ϵ-greedy policy. We store the resulting state, action and reward information in an experience replay buffer. We can then update the DQN weights with the following loss,

$$\mathcal{L}(\omega) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim \mathcal{D}}\left[(Y - Q_\omega(\mathbf{s}_t, \mathbf{a}_t))^2\right], \tag{2.15}$$

where the expectation in equation (2.15) is using samples from some dataset $\mathcal{D}$. One problem in this loss is that the targets are also dependent on the parameters ω, which could make the optimization unstable. As a solution, we can maintain another target Q-function $Q_{\omega_{targ}}$ that is very close to Q_ω but lags behind it. The parameters of this network are updated using Polyak averaging [150],

$$\omega_{targ} \leftarrow \rho\, \omega_{targ} + (1 - \rho)\omega. \tag{2.16}$$

The DQN algorithm can be used successfully for continuous and large-dimensional state spaces. However, it cannot handle continuous or large-dimensional action spaces, since the policy still needs to select actions based on equation (2.13).

2.3 POLICY-BASED METHODS

Unlike value-based methods, policy-based RL approaches directly attempt to find the optimal policy instead of deriving it based on the value function. Policy-based RL methods share many similarities with the dynamic programming policy iteration (PI) algorithm. PI is an iterative algorithm that is guaranteed to converge to the optimal policy

for finite MDPs. Starting from arbitrary initial value estimates and (deterministic) policy, PI iterates two steps until the policy converges. The first step is policy evaluation. In this step, we iteratively update the values of each state,

$$V(\mathbf{s}_t) \leftarrow \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \pi(\mathbf{s}_t))[r(\mathbf{s}_t, \pi(\mathbf{s}_t)) + \gamma V(\mathbf{s}_{t+1})]. \quad (2.17)$$

This step is repeated until the maximum state value divergence is below a predefined threshold. Given the results of policy evaluation, a PI iteration proceeds with to the policy improvement (second step). This step updates the policy for each state,

$$\pi(\mathbf{s}_t) \leftarrow \operatorname*{argmax}_{\mathbf{a}_t \in \mathcal{A}} \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)[r(\mathbf{s}_t, \mathbf{a}_t) + \gamma V(\mathbf{s}_{t+1})]. \quad (2.18)$$

2.3.1 *Policy Gradient*

For non-finite MDPs, we can use a parametrized policy. At the center of the policy-based RL methods for parametrized policies is the policy gradient (PG) algorithm. This algorithm optimizes policy parameters with respect to the RL objective from equation (2.1). This is done by computing the gradient of this object with respect to those parameters. This value $\nabla_\delta J(\pi)$ is referred to as the policy gradient. Following [205], we can derive a tractable estimator of this value,

$$\begin{aligned} \nabla_\delta J(\pi) &= \nabla_\delta \mathbb{E}_{\tau \sim \pi_\delta}[R(\tau)] && (2.19)\\ &= \nabla_\delta \int p(\tau \mid \delta) R(\tau) d\tau && (2.20)\\ &= \int \nabla_\delta p(\tau \mid \delta) R(\tau) d\tau. && (2.21) \end{aligned}$$

Based on the log-derivative and the chain rule, one could show that

$$\nabla_\delta p(\tau \mid \delta) = p(\tau \mid \delta) \nabla_\delta \log p(\tau \mid \delta). \quad (2.22)$$

Plugging this into equation (2.21), we obtain

$$\begin{aligned} \nabla_\delta J(\pi) &= \int p(\tau \mid \delta) \nabla_\delta \log p(\tau \mid \delta) R(\tau) d\tau && (2.23)\\ &= \mathbb{E}_{\tau \sim \pi_\delta}[\nabla_\delta \log p(\tau \mid \delta) R(\tau)]. && (2.24) \end{aligned}$$

The trajectory distribution can be fully captured using the initial state distribution, the transition dynamics, and the policy. We can rewrite $\nabla_\delta \log p(\tau \mid \delta)$,

$$\nabla_\delta \log p(\tau \mid \delta) = \nabla_\delta \log \left[\rho_0(\mathbf{s}_0) \prod_{t=0}^{T} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \pi_\delta(\mathbf{a}_t \mid \mathbf{s}_t) \right] \tag{2.25}$$

$$\begin{aligned} &= \nabla_\delta \log \rho_0(\mathbf{s}_0) + \sum_{t=0}^{T} [\nabla_\delta \log p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \\ &\quad + \nabla_\delta \log \pi_\delta(\mathbf{a}_t \mid \mathbf{s}_t)]. \end{aligned} \tag{2.26}$$

Since the initial state distribution and environment dynamics do not directly influence δ, we can finally write

$$\nabla_\delta \log p(\tau \mid \delta) = \sum_{t=0}^{T} \nabla_\delta \log \pi_\delta(\mathbf{a}_t \mid \mathbf{s}_t). \tag{2.27}$$

If we plug this into equation (2.24), we obtain

$$\nabla_\delta J(\pi) = \mathbb{E}_{\tau \sim \pi_\delta} \left[\sum_{t=0}^{T} \nabla_\delta \log \pi_\delta(\mathbf{a}_t \mid \mathbf{s}_t) R(\tau) \right]. \tag{2.28}$$

In practice, we can use a mean estimator to compute this gradient using trajectories collected by following the policy. This algorithm can suffer from high variance, making it less stable and sample-inefficient. We can significantly reduce this effect by replacing the returns with the advantage function defined as

$$A^{\pi_\delta}(\mathbf{s}, \mathbf{a}) = Q^{\pi_\delta}(\mathbf{s}, \mathbf{a}) - V^{\pi_\delta}(\mathbf{s}). \tag{2.29}$$

This function measures how good it is to take action $\mathbf{a}$ in state $\mathbf{s}$, compared to an action obtained by random sampling of the policy π_δ. We can replace the returns with the advantages A,

$$\nabla_\delta J(\pi) = \mathbb{E}_{\tau \sim \pi_\delta} \left[\sum_{t=0}^{T} \nabla_\delta \log \pi_\delta(\mathbf{a}_t \mid \mathbf{s}_t) A^{\pi_\delta}(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{2.30}$$

This modification removes the effect of rewards from previous steps (when using returns) and reduces variance while keeping the expectation unbiased [181, 182]. In practice, we estimate the advantages by learning Q^{π_δ} and V^{π_δ} using the methods introduced in section 2.2. In fact, learning V^{π_δ} is sufficient for estimating the advantages since Q^{π_δ}, can be replaced by the reward-to-go. This is an implementation detail that differs from one implementation to another. The PG algorithm laid the foundation for a family of other policy-based methods. These methods attempt to alleviate some of the disadvantages of this method such as its large variance and being on-policy. Despite its shortcomings, this algorithm has also been used for robotics manipulation [148].

2.3.2 *Trust Region Policy Optimization*

Large policy updates could destabilize policy optimization. Therefore, there is a clear need to limit those updates without hindering the learning process. Schulman et al. [165] proposed an approach to achieve this by framing the problem as a constraint optimization. This method establishes a trust region around the old policy during policy updates. Hence, it is called trust region policy optimization (TRPO). TRPO proposes a surrogate objective to approximate the PG objective. This objective extends the lower bound on the expected discounted reward, introduced in [88], to stochastic policies.

$$\mathcal{L}(\delta_{old}, \delta) = \mathbb{E}_{\tau \sim \pi_{\delta_{old}}} \left[\frac{\pi_\delta(\mathbf{a} \mid \mathbf{s})}{\pi_{\delta_{old}}(\mathbf{a} \mid \mathbf{s})} A^{\pi_{\delta_{old}}}(\mathbf{s}, \mathbf{a}) \right], \tag{2.31}$$

where $\pi_{\delta_{old}}$ is the (fixed) current policy before the update. Notice that this expression uses importance sampling to weigh the past actions under the old policy based on how likely they are to be taken under the new policy. Furthermore, TRPO imposes a constraint on the policy change using the average Kullback-Leibler divergence (KL) between the policies. We can now write the optimization problem,

$$\begin{aligned} &\underset{\delta}{\operatorname{argmax}} \mathcal{L}(\delta_{old}, \delta) \\ &\text{s.t.} \quad \mathbb{E}_{s \sim \pi_{\delta_{old}}} [\mathrm{KL}(\pi_\delta(. \mid \mathbf{s}) \parallel \pi_{\delta_{old}}(. \mid \mathbf{s}))] \leqslant \zeta, \end{aligned} \tag{2.32}$$

where ζ is a scalar hyperparameter. In TRPO, this optimization problem is approximately solved using the conjugate gradient method followed by a backtracking line search. The conjugate gradient method alleviates the need to directly compute and store the fisher information matrix of the KL, which is needed for the solution of the Taylor-expansion-based approximation of the optimization problem in equation (2.32). More details can be found in the original publication [165]. Due to the complexity of this optimization problem, TRPO updates can be expensive in practice. TRPO is a very popular algorithm and has been used in the past for robotics use cases [81].

2.3.3 *Proximal Policy Optimization*

Although TRPO successfully illustrates the benefits of limiting policy updates for policy gradient methods, it increases computational complexity by means of trust region optimization. The computational cost can be reduced by limiting the number of conjugate gradient iterations. However, this comes at the cost of increasing the approximation gap. Schulman et al. [166] propose some tricks to achieve the same goal while reducing the computational cost. They propose an algorithm called proximal policy optimization (PPO). Instead of imposing a trust region on the objective in equation (2.31), PPO proposes a sim-

ple modification of the objective. It reduces large policy updates by clipping the objective function,

$$\mathcal{L}(\delta_{old}, \delta, \epsilon) = \mathbb{E}_{\tau\sim\pi_{\delta_{old}}}\big[\min\big(r(\delta)A^{\pi_{\delta_{old}}}(\mathbf{s},\mathbf{a}), \\ \text{clip}(r(\delta), 1-\epsilon, 1+\epsilon)A^{\pi_{\delta_{old}}}(\mathbf{s},\mathbf{a})\big)\big], \tag{2.33}$$

where $r(\delta) = \pi_\delta(\mathbf{a} \mid \mathbf{s})/\pi_{\delta_{old}}(\mathbf{a} \mid \mathbf{s})$ and ϵ is a hyperparameter. This simple change removes the incentive for the rate of policy change, as quantified by $r(\delta)$, to be outside of the interval $[1-\epsilon, 1+\epsilon]$. The clipping effectively removes any gradients that push in that direction. This makes the degree of policy change controllable at the cost of introducing a new hyperparameter. More importantly, this change does not introduce any additional computational complexity to vanilla PG. PPO is a very popular algorithm in robotics, specifically for transferring policies from simulation to the real world [3, 38, 69, 183, 184].

2.3.4 *Twin-Delayed Deep Deterministic Policy Gradients*

So far, we have only looked at policy-based methods which can learn using trajectories collected using the agent's own policy. Methods of this sort are called on-policy RL algorithms. In contrast, some of the value-based approaches we have examined, such as (deep) Q-learning, could benefit from data collected using another policy. Methods that have this capability are called off-policy RL algorithms. In practice, this property means that these methods could also benefit from environment interactions that were performed by older iterations of the agent's policy or even another agent's policy (e.g., data collected by a human demonstrator). Due to the high sample requirements of RL methods, this property is very desirable. This is especially the case for real-world robotics, where the environment dynamics and control can be very complex. Hence, there is a clear need for off-policy algorithms that can handle continuous action spaces. In section 2.2.2 we discarded the usage of DQN for continuous action spaces due to the intractability of the maximum operation in equation (2.13). However, it is possible to use the learned Q-function to optimize a parametrized policy. deep deterministic policy gradient (DDPG) [116] is an algorithm that follows this paradigm. Given a deterministic policy $\pi_\delta(\mathbf{s})$, and a Q-function Q_ω learned using a version of the loss in equation (2.15) that uses the parametrized policy when computing the DQN targets,

$$Y = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q_\omega(\mathbf{s}_{t+1}, \pi_\delta(\mathbf{s}_{t+1})), \tag{2.34}$$

DDPG learns a policy by maximizing the expected Q-values,

$$\max_\delta \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[Q_\omega(\mathbf{s}, \pi_\delta(\mathbf{s}))], \tag{2.35}$$

where $\mathcal{D}$ is a dataset of experiences collected by the agent using all the policy versions it has learned up until now. This dataset is also known as the experience replay buffer. Hence, DDPG is considered an off-policy algorithm, benefiting from old experiences to improve its sample efficiency. However, in practice, DDPG can suffer from the overestimation of the Q-values, which is then negatively exploited during policy optimization [54]. To address this issue, Fujimoto, Hoof, and Meger [54] proposed a couple of remedies. First, they add noise to the policy actions in equation (2.35), to discourage the policy optimization from exploiting these overestimations. This modified policy is noted $\pi'_\delta(\mathbf{s})$. Second, they propose maintaining two Q-functions Q_{ω_1} and Q_{ω_2} that are each learned as shown in equation (2.15). When forming the target, they use the minimum Q-value ,

$$Y = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \min_{i=1,2} Q_{\omega_i}(\mathbf{s}_{t+1}, \pi'_\delta(\mathbf{s}_{t+1})). \tag{2.36}$$

Lastly, Fujimoto, Hoof, and Meger [54] recommends updating the Q-functions more frequently than the policy itself. The resulting algorithm is called twin delayed deep deterministic policy gradient (TD3). Both DDPG and TD3 are very popular algorithms in the RL literature, and have also been used for robotic manipulation tasks [127, 152].

2.3.5 *Soft Actor-Critic*

Real-world robot control suffers from noisy sensors and actuators, dynamic environments, communication delays, modeling inaccuracies, and many other factors that contribute to the presence of uncertainty. This motivates the development of algorithms for learning stochastic policies for such systems. Haarnoja et al. [64] proposed an off-policy RL algorithm for learning stochastic policies. The algorithm is called soft actor-critic (SAC). SAC augments the RL objective from equation (2.1) with an entropy maximization term [222],

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T-1} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) + \kappa H(\pi(. \mid \mathbf{s}_t))\right], \tag{2.37}$$

where κ is hyperparameter that controls the effect of the entropy term H on the total reward. This modification to the objective encourages stochastic policy and increases exploration. In practice this is implemented by integrating entropy maximization into the action-value function, to form the soft Q-function,

$$\begin{aligned} Q(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}\sim p(.|\mathbf{s}_t,\mathbf{a}_t)}\big[r(\mathbf{s}_t, \mathbf{a}_t) \\ + \gamma \mathbb{E}_{\mathbf{a}_{t+1}\sim\pi(.|\mathbf{s}_{t+1})}[Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) + H(\mathbf{s}_{t+1})]\big]. \end{aligned} \tag{2.38}$$

Learning this Q-function is done by modifying the targets,

$$Y = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\big(\min_{i=1,2} Q_{\omega_i}(\mathbf{s}_{t+1}, \pi'_\delta(\mathbf{s}_{t+1})) + H(\mathbf{s}_{t+1})\big). \tag{2.39}$$

Notice that SAC also uses two Q-function approximators as in [54]. The policy update step in SAC pushes the policy towards the exponential of the soft Q-functions. This can be done with the following policy loss,

$$\mathcal{L}(\delta) = \mathbb{E}_{\mathbf{s}\sim\mathcal{D}}\left[\mathrm{KL}\left(\pi_\delta(.\mid\mathbf{s}) \,\middle\|\, \frac{Q_\omega(\mathbf{s},.)}{Z_\omega(\mathbf{s})}\right)\right], \tag{2.40}$$

where $Z_\omega(\mathbf{s})$ is the partition function that normalizes the distribution. Practical implementations of SAC use the reparametrization trick to sample from π_δ, the loss can then be written as

$$\mathcal{L}(\delta) = \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\epsilon\sim\mathcal{N}}[\log\pi_\delta(f_\delta(\epsilon,\mathbf{s})\mid s) - Q_\omega(\mathbf{s}, f_\delta(\epsilon,\mathbf{s}))], \tag{2.41}$$

where ϵ is a random noise variable sampled from a Gaussian distribution $\mathcal{N}$, and $f_\delta(\epsilon,\mathbf{s})$ is the reparametrized policy. This reparametrization reduces the variance in the loss estimation. SAC is a relatively sample-efficient algorithm and has been used over the years for multiple robot control applications [10, 64, 84].

Part II

ROBOT STATE REPRESENTATIONS

This part of the thesis is based on ideas, theories, and experiments that appeared in the following publications:

Elie Aljalbout, Maximilian Ulmer, and Rudolph Triebel. "Seeking visual discomfort: Curiosity-driven representations for reinforcement learning." In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 3591–3597. DOI: 10.1109/ICRA46639.2022.9811663.

Axel Sauer*, Elie Aljalbout*, and Sami Haddadin. "Tracking holistic object representations." In: *British Machine Vision Conference (BMVC)*. 2019.

Direct quotes from the papers are highlighted in gray font color. Sections or subsections with titles marked with ‡ correspond to (sub-)sections from the author's corresponding previously published papers.

Supplementary material related to this part's chapters is collected in Appendix A.

3

SUPERVISED AND SELF-SUPERVISED LEARNING OF STATE REPRESENTATIONS

To manipulate their environments, robots need first to understand their state and the state of their surroundings. This is the robot perception problem. Specifically, robots are equipped with sensors that provide them with observations as feedback about their state and the state of their world. Understanding these observations is crucial to planning their actions and controlling their bodies. This is a challenging problem since sensor measurements can be very complex to process, high-dimensional, and noisy. In this part of the thesis, we will propose supervised and self-supervised approaches to learning state representations that are beneficial for robotics. This chapter will first introduce the problem and some background on the available methodology and state-of-the-art.

3.1 PROBLEM DEFINITION

At each time step t, the agent receives observations $\mathbf{o}_t$ from its different sensors. To use these measurements for control, the agent must extract a state $\mathbf{s}_t$ that describes its configuration, as well as the state of the environment. This corresponds to extracting all control-relevant information from $\mathbf{o}$ and representing them in a format suitable for downstream planning and control. The representation problem is common in multiple aspects of robotics. For example, for robot control, different control representations (e.g., configuration or task space) exist for achieving the same goal. These different control representations also affect perception. Depending on the selected control space, different state representations can be beneficial. For instance, configuration space control requires joint-level feedback from the robots but does not necessarily require feedback about the Cartesian positions of the different links. In addition, the same control space could require different state representations depending on the task at hand. For example, if the robot is performing a 6-dimensional manipulation task, all dimensions of the robot's and the environment states are required. The same does not necessarily apply to a planar task. One could possibly include all available information in all its different representations in the state representations. However, such an approach would result in a large state space and would eventually complicate the process of policy search needed to derive the downstream planners and controllers with reinforcement learning. Therefore, we could define the problem as follows: Given the observations $\mathbf{o}_t$ extract the most compact state representation $\mathbf{s}_t$ that contains the neces-

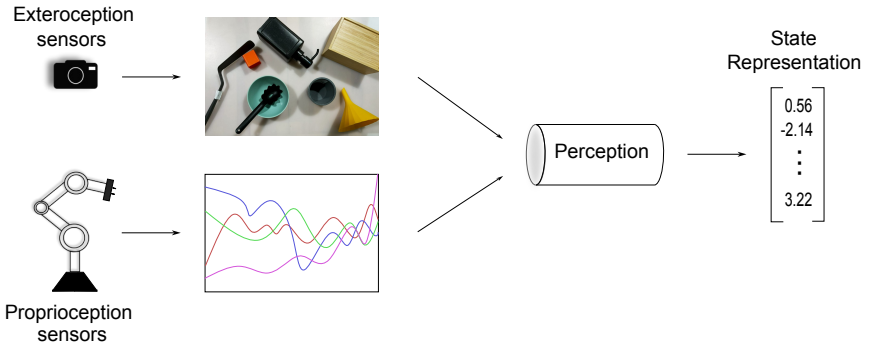


Figure 3.1: Given measurements from proprioceptive and exteroceptive sensors, a perception module should extract a compact state representation that can be used for downstream control. The state representation can contain explicit physical values, but can also have implicit information useful for control in a non-interpretable format.

sary and sufficient information for downstream planning and control. This is illustrated in Figure 3.1. This illustration does not consider the sequential nature of real-world observations. Instead, it depicts the most simplified and naive inference process that considers each state to be fully recoverable based on the current observation of the environment and independent of previous states, observations, or actions. In addition to compactness, a state representation is desired to have other properties, such as sparsity, being Markovian, smoothness, natural clustering, temporal and spatial coherence, as well as simplicity, and disentanglement of latent factors [20, 110].

3.2 INFERRING STATES FROM OBSERVATIONS

In the classical approach to robotics, the state of the system can be inferred on the basis of elaborate perception pipelines designed for each task and each robot. For example, to push an object around a table, a robot needs access to the pose and velocity of that object in some predefined coordinate system. Such information can be inferred from camera observations via a combination of object tracking and filtering methods. Different components of such a pipeline (in our running example: tracking and filtering algorithms) can be designed using established principles of multiple-view geometry as well as filtering techniques [122]. Alternatively, such components could also be learned using supervised machine learning methods.

More recently, in RL-centric approaches to robotics, state representations can be learned in a self-supervised manner that relies only on data containing variables that can be observed by the agent and no additional labels. This paradigm brings forward the promise of having one single algorithm that could be applied to different tasks and different robots. It does so by reducing the algorithmic assumptions

regarding the availability and robustness of perception components or labeled datasets to obtain them. Instead, these methods solely rely on the data collected by the RL agent while exploring its environment.

It is important to note that in certain cases the exact state of the environment is not recoverable from the available sensory measurements. The environment is then said to be partially observable. In such a case, the goal of perception is to extract the most valuable information for control and represent it in a way that facilitates learning the downstream tasks.

3.3 PERCEPTION MODALITIES

Depending on the task at hand, different sensor modalities can be used for perception. However, one modality, vision, has been shown to be central to the interaction of humans and various animals with their environments. In artificial agents, vision can be implemented using cameras and algorithms to process images. This problem is very challenging since camera images can be very high-dimensional, blurry, and complex to process. However, the information contained in images is necessary for most robot manipulation tasks. Replacing vision would require a costly combination of multiple other sensing modalities. Hence, this part of the thesis mostly focuses on visual perception and introduces supervised and self-supervised methods for obtaining state representation based on vision-centric observations. Some of the discussed methods — especially when based on unsupervised learning — can also be used for extracting state representations from other sensory inputs.

3.4 SUPERVISED LEARNING OF STATE REPRESENTATIONS

3.4.1 *Template-matching*

In certain cases, datasets of sensor measurements and corresponding state-related labels can be available or collected for a certain task. In such a case, one could use supervised learning methods to learn a direct mapping from observations to states. For a robot to manipulate objects in the world, it would first need to locate those objects at each step. This defines the visual object tracking problem. More explicitly, tracking starts with initial object detections which are used to assign different identifiers for separate objects. From there on, in each new camera frame, a tracker looks for the position of a given object (associated with an identifier) in that new frame.

Numerous methods have been proposed to address the challenge of visual tracking. The next chapter will present a method developed during this thesis for constructing holistic multi-template mod-

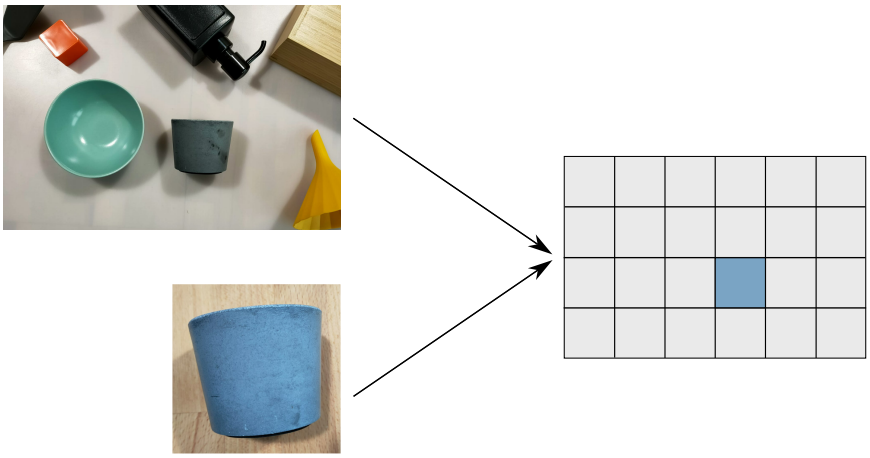


Figure 3.2: An illustration of template matching: Given an input image and an image template, template matching returns a map of the occurrences of the template object in the input image. The map is usually given at multiple resolutions to enable more accurate results. This figure illustrates the concept with a single resolution for the sake of simplicity.

ules. Hence, this section will provide a brief overview of template-matching techniques for tracking. Additionally, it will discuss Siamese network-based methods, which were also leveraged in the proposed method.

Template matching methods are very popular methods in visual tracking due to their notable advantages in terms of speed and accuracy. In essence, these methods involve using a template image of the target object and attempting to locate the object by matching the template to regions within the given image and finding the most similar patches. The process is illustrated in Figure 3.2. More formally, template matching computes a similarity matrix where each entry $S_{i,j}$ corresponds to the similarity score of the corresponding image patch $I_{i,j}$ to the template image I_t,

$$S_{i,j} = k(\Phi_{tp}(I_t), \Phi_{tg}(I_{i,j})), \tag{3.1}$$

where k is an arbitrary similarity measure, Φ_{tp}, and Φ_{tg} are a feature extractors. The actual functions used depend on the specific application. Given this matrix, it is straightforward to extract patches that mostly resemble the target object and then choose which actually correspond to it based on some predefined confidence requirement. To account for the object having different scales in the target and template images, multiple scales of the template images are processed. The matching technique can be used to compare image intensities (using metrics such as normalized cross-correlation or the sum of squared differences), gradient features [132], or any other applicable features for the tracking task [77, 138, 174].

3.4.2 *Siamese trackers*

Computing similarity measures in the pixel space of images could lead to a non-representative metric. Therefore, there is a clear need for image feature extractors on which template matching can be applied. In recent years, convolutional Siamese neural networks have become very popular feature extractors for tracking [24, 113, 114, 185]. As their name suggests, Siamese networks use the same network (parameters and weights) to process different inputs representing different entities. For example, when used as feature extractors during template matching, they would correspond to setting $\Phi_{tp} = \Phi_{tg}$ in equation (3.1). Siamese Instance Search Tracking [185] trains a Siamese network with the margin contrastive loss to extract features. These features are then used to compare the target image with patches taken from the area around the previously detected image location. The patch with the highest score is then identified as the match. Unlike the work in [185], Bertinetto et al. [24] propose "fully convolutional Siamese networks", a method that employs fully convolutional networks to eliminate the bias towards the middle part of the image. In addition, their method utilizes the embedding of the template as the correlation filter for the search image, enabling real-time performance. Li et al. [114] proposed an extension of this technique, which uses region proposal networks [159] to extract proposals from the correlation feature maps. In addition to the architecture in [24], they also incorporate a bounding box regression branch similar to the one used in [185]. In order to tackle the discrepancy between the amount of positive and negative samples during training, Zhu et al. [221] suggests the use of distractor objects from previous patches as negative samples for current ones. This augmentation enhances the feature representation to more accurately differentiate target objects from similar distractors. In contrast to the methods mentioned above, Held, Thrun, and Savarese [71] proposed an approach that does not require patch sampling and only needs the search image and a patch of the current image (centered around the old detection) to predict the position of the bounding box. This approach is faster and can handle changes in aspect ratio and scale; however, its accuracy is not as good as that of contemporary methods.

3.4.3 *Recent trends in object tracking*

Recent approaches to visual object tracking have shifted away from Siamese template matching and have increasingly adopted concepts from generative machine learning. With the success of transformers as sequence models, their application to tracking was natural. For instance, Cui et al. [43] proposes an end-to-end tracking method that uses an iterative mixed attention mechanism to capture both spatial and temporal information. Wei et al. [201] proposes a novel autore-

gressive approach for visual tracking that uses a transformer-based architecture. Similarly, Chen et al. [40] propose a sequence-to-sequence learning approach for visual object tracking that uses a transformer-based architecture. Gao et al. [57] propose an attention-in-attention module to improve the attention mechanism for transformers used in visual tracking. The proposed module can effectively enhance the appropriate correlations and suppress erroneous ones by seeking consensus among all correlation vectors. In addition to transformers, masked autoencoding has also been used in recent methods [39, 207]. The work in [39] proposes a simplified architecture for visual object tracking that uses masked autoencoders. Furthermore, Wu et al. [207] propose a simple extension of masked autoencoders pretraining on videos for matching-based downstream tasks, including visual object tracking and video object segmentation. Finally, recent work has shown that language can also be beneficial for tracking [52, 210]. For instance, Feng et al. [52] argue that conditioning on the natural language description of a target provides information for longer-term invariance and thus helps cope with typical tracking challenges. The paper proposes a novel deep tracking-by-detection formulation that can take advantage of language descriptions. Similarly, Yan et al. [210] propose UNINEXT, a model that can perform diverse instance perception tasks by reformulating them into a unified object discovery and retrieval paradigm. UNINEXT can handle different types of objects by changing the input prompts, and can exploit data from different tasks and label vocabularies for joint training of general instance-level representations

3.5 SELF-SUPERVISED LEARNING OF STATE REPRESENTATIONS

In many scenarios, it is not possible to obtain a dataset of sensor observation and the corresponding state-related labels. Collecting such datasets is a very expensive and time-consuming process since state-of-the-art perception methods tend to rely on very large amounts of data to be trained. Simultaneously, it is very accessible to obtain unlabeled sensory measurements such as images just from the Internet. Furthermore, while learning to perform a task by trial and error, a robot can easily collect a dataset of sensory measurements (without state labels). Such datasets could contain observations that have a distribution similar to the ones seen at inference time. Therefore, being able to learn mappings from observations to state representations from such unlabeled data would greatly benefit robot perception. However, this lack of labels makes the learning process more complex and inefficient. On the positive side, it removes the human bias from the data since human-labeled data often follow some explicit state-space design. For example, when designing the state space of some task, a human would typically focus on the positional information of the agent and objects and omit information about its shape,

object type, or other physical properties. The latter are hard to label, but potentially inferrable from the observations and beneficial to the task.

Self-supervised state representation learning (SRL) approaches can either be discriminative or generative. Each class of methods offers distinct strategies for feature extraction without the need for explicit annotations or labels. The key distinction between discriminative and generative self-supervised representation learning methods lies in their learning objectives. Discriminative methods aim to capture discriminative features that are directly relevant to a specified task, while generative methods focus on capturing the data distribution and underlying data manifold.

3.5.1 *Discriminative approaches*

One of the most popular approaches to state representation learning is the autoencoder (AE) [16]. AEs are neural networks designed to learn compact representations of data by mapping it into a latent space (typically with lower dimension) and then reconstructing the original data from this compressed representation. The network consists of an encoder f_ψ that maps the input data $\mathbf{o}$ to a latent representation $\mathbf{z}$, and a decoder g_θ that reconstructs the data from the encoded representation. Mathematically, an autoencoder can be represented as

$$\mathbf{z} = f_\psi(\mathbf{o}), \tag{3.2}$$

$$\hat{\mathbf{o}} = g_\theta(\mathbf{z}), \tag{3.3}$$

where $\mathbf{o}$ represents the raw input observation, $\mathbf{z}$ is the latent state representation, and $\hat{\mathbf{o}}$ is the reconstructed input: AEs are trained using a mean squared error (MSE) loss between the input and reconstructed data based on a dataset $\mathcal{D}$, using the following loss function,

$$\mathcal{L}_{AE}(\mathcal{D}, \theta, \psi) = \mathbb{E}_{\mathbf{o}\sim\mathcal{D}}\left[\mathrm{MSE}(\mathbf{o}, g_\theta(f_\psi(\mathbf{o}))\right]. \tag{3.4}$$

Lange and Riedmiller [104] proposed one of the first methods to integrate AEs in batch-RL. Later work explored the use of regularized autoencoder (RAE) [212]. RAEs are trained using a very similar loss to traditional autoencoders,

$$\mathcal{L}_{RAE}(\mathcal{D}, \theta, \psi) = \mathbb{E}_{\mathbf{o}\sim\mathcal{D}}\left[\mathrm{MSE}(\mathbf{o}, g_\theta(f_\psi(\mathbf{o})) + \lambda_z \;\middle|\; |\mathbf{z}\|^2 + \lambda_\theta \|\theta\|^2\right]. \tag{3.5}$$

In addition to input reconstruction, this loss explicitly penalizes the learned representation $\mathbf{z} = f_\psi(\mathbf{o})$ and the decoder weights θ. λ_z and λ_θ are scalar terms that weigh the effect of the previously mentioned regularizers. AEs could either be trained simultaneously with the policy [35, 212], or in certain cases, separately pretrained before the RL phase begins [4, 74, 108] or in an alternating fashion [104]. A similar class of methods simultaneously learns feature encoders and predictive models [14, 59]. For example, [14] learns a low-dimensional

feature embedding of images jointly with a forward dynamics model in this low-dimensional feature space. This approach is efficient and enables end-to-end learning from pixels to torques. Similarly, inverse dynamics prediction [214] and reward prediction [134] can be used as auxiliary tasks for learning state representations.

Another popular paradigm for SRL is contrastive learning (CL). CL is a technique that aims to learn low-dimensional representations of data by contrasting similar and dissimilar samples. Specifically, it tries to bring similar samples close to each other in the representation space and push dissimilar samples far apart using some similarity metric, such as the Euclidean distance. Popular methods for contrastive learning include Siamese networks [42, 99], triplet losses [76, 198], and noise-contrastive estimation [139]. One popular CL objective $\mathcal{L}_{CL}$ is the InfoNCE loss [139], computed as

$$\mathbb{E}_{q,k_+,\{k_i\}^K\sim\mathcal{D}}\left[\log \frac{\exp f_\psi(q)^T W f_\psi(k_+)}{\exp f_\psi(q^T) W f_\psi(k_+) + \sum_{i=0}^{K-1} f_\psi(q)^T W f_\psi(k_i)}\right], \tag{3.6}$$

where q is the anchor, k_+ is the positive sample, $\{k_i\}^K$ are the negatives samples, and W is a matrix used for the bilinear product operation. When used in an RL framework, the choice of negative and positive sample should be made based on the data in the replay buffer. Laskin, Srinivas, and Abbeel [105] use data augmentations as positive samples and all other samples in the batch, as well as their augmentations as negative ones. Similarly, the work in [175] uses contrastive learning to associate pairs of observations separated by a short time difference, hence uses (near) future observations as positive queries and all other samples in the batch as negative ones.

Both AE-based methods and contrastive learning focus on compression of observation as the main goal for SRL. Besides autoencoding and contrastive methods, it is possible to integrate any other self-supervised or unsupervised objective for SRL, such as predictive modeling [108], clustering [6, 66], or masked reconstruction [70]. For example, Jonschkowski and Brock [86] presented an approach for SRL based on enforcing physical properties such as proportionality, causality, repeatability, and temporal coherence. These properties are formulated as objectives on the latent representations and are called robotic priors.

3.5.2 *Generative Approaches*

By modeling data distributions, generative approaches can learn latent state representations that are fit for generating new data but also for downstream tasks such as control. variational autoencoders (VAE) [98, 160] have been shown to be very useful for this purpose. VAEs differ from traditional autoencoders in several key ways. Firstly,

VAEs introduce a probabilistic approach to encoding and decoding, modeling the latent space as a probability distribution, whereas traditional AEs focus solely on deterministic mappings. Secondly, VAEs enable sampling from the latent space, allowing for the generation of novel data points. Additionally, VAEs incorporate a regularization term in their loss function, promoting the learning of structured and continuous latent representations. This probabilistic nature and regularization make VAEs more adept at data generation, representation learning, and tasks involving uncertainty.

The VAE framework is based on the concept of maximizing the evidence lower bound (ELBO), also known as the variational lower bound. The ELBO is a fundamental equation in variational inference and, in the context of VAEs, it serves as the objective function for training. Next, we will show how the ELBO is derived in the original publications [98, 160]. First, we assume that the observed data points $\mathbf{o}$ are generated based on a latent variable $\mathbf{z}$. This means that $\mathbf{z}$ encapsulate the fundamental nature and meaning of data. The joint distribution of $\mathbf{o}$ and $\mathbf{z}$ can be written as

$$p(\mathbf{o}, \mathbf{z}) = p(\mathbf{o} \mid \mathbf{z})p(\mathbf{z})), \tag{3.7}$$

where $p(\mathbf{z})$ and $p(\mathbf{o} \mid \mathbf{z})$ are the prior and likelihood distributions, respectively. While the prior defines the structure of the latent variables, the likelihood distribution captures the process by which the latent variable is mapped into an observation. We can obtain $p(\mathbf{o})$ by marginalizing the joint distribution,

$$p(\mathbf{o}) = \int p(\mathbf{o}, \mathbf{z}) d\mathbf{z} = \int p(\mathbf{z})p(\mathbf{o} \mid \mathbf{z}) d\mathbf{z}. \tag{3.8}$$

Assuming we have a process by which we can sample from $p(\mathbf{z})$, using Monte Carlo Integration to estimate this value becomes intractable for continuous latent spaces (or discrete ones with a large dimension). Alternatively, we can then express $p(\mathbf{o})$ as follows:

$$p(\mathbf{o}) = \frac{p(\mathbf{o} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{o})}. \tag{3.9}$$

However, $p(\mathbf{z} \mid \mathbf{o})$ is typically not accessible and hence this expression cannot be evaluated. Variational inference offers us the means to approximate the true posterior $p(\mathbf{z} \mid \mathbf{o})$ with a variational distribution $q(\mathbf{z} \mid \mathbf{o})$. In the case of VAEs, we use a density network to represent this distribution as $q_\psi(\mathbf{z} \mid \mathbf{o})$. The latter is often referred to as a recognition model. From the classical AE perspective, this network is the

encoder. We will now examine the KL between the approximate and the true posterior,

$$\mathrm{KL}\big(q_\psi(\mathbf{z} \mid \mathbf{o}) \,\big|\big|\, p(\mathbf{z} \mid \mathbf{o})\big) = \mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}\big[\log q_\psi(\mathbf{z} \mid \mathbf{o}) - \log p(\mathbf{z} \mid \mathbf{o})\big] \tag{3.10}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}\left[\log q_\psi(\mathbf{z} \mid \mathbf{o}) - \log \frac{p(\mathbf{o} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{o})}\right] \tag{3.11}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}[\log q_\psi(\mathbf{z} \mid \mathbf{o}) - \log p(\mathbf{z}) - \log p(\mathbf{o} \mid \mathbf{z})] + \log p(\mathbf{o}). \tag{3.12}$$

By rearranging the terms, we obtain

$$\log p(\mathbf{o}) = \mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}[\log p(\mathbf{o} \mid \mathbf{z}) + \log p(\mathbf{z}) - \log q_\psi(\mathbf{z} \mid \mathbf{o})] + \mathrm{KL}\big(q_\psi(\mathbf{z} \mid \mathbf{o}) \,\big|\big|\, p(\mathbf{z} \mid \mathbf{o})\big). \tag{3.13}$$

Since the KL is always positive, the expectation on the right-hand side of equation (3.13) is a lower bound to $\log p(\mathbf{o})$. This term is the variational lower bound. Also, since $\log p(\mathbf{o})$ is referred to as evidence, the variational lower bound is more commonly referred to as the evidence lower bound. Hence, VAEs also parameterize a generative model $p_\theta(\mathbf{o} \mid \mathbf{z})$ and can be trained using the objective

$$\underset{\theta,\psi}{\operatorname{argmax}}\Big(\mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}[\log p_\theta(\mathbf{o} \mid \mathbf{z})] - \mathrm{KL}\big(q_\psi(\mathbf{z} \mid \mathbf{o}) \,\big|\big|\, p(\mathbf{z})\big)\Big). \tag{3.14}$$

The first term in equation (3.14) is a data reconstruction term, and the second one is the KL between the prior and the posterior, effectively acting as a regularizer. Optimizing with respect to ψ involves computing a gradient of the form

$$\nabla_\psi \mathbb{E}_{\mathbf{z}\sim q_\psi}[f(\mathbf{z})], \tag{3.15}$$

which can be estimated using algorithms such as REINFORCE [205]. Such an optimization scheme would suffer due to the large variance in the gradient estimation. Instead, VAEs are typically trained using the reparametrization trick, where the sampling process is expressed as a deterministic function of the distribution parameters and a random sample ϵ drawn from some base distribution $p(\epsilon)$. Conveniently, VAEs employ Gaussian distributions for both recognition and generative models. In practice, this is implemented using a neural network that outputs a mean and standard deviation. For instance, $q_\psi(\mathbf{z} \mid \mathbf{o})$ would look like

$$[\mu_z, \sigma_z] = \mathrm{NN}_\psi(\mathbf{o}), \tag{3.16}$$

$$q_\psi(\mathbf{z} \mid \mathbf{o}) = \mathcal{N}\big(\mathbf{z}; \mu_z, \sigma_z^2\big). \tag{3.17}$$

In the RL literature, VAEs are commonly integrated in the learning process as a way to learn perception based on data collected from the same RL environment. Higgins et al. [74] introduced one of the earliest methods to follow this paradigm. Such methods [4, 192] often employ a variation of variational autoencoders called β-VAE [73]. The main difference between the two methods is a that β-VAEs introduce a scaling factor to the KL term in equation (3.14),

$$\underset{\theta,\psi}{\operatorname{argmax}}\Big(\mathbb{E}_{\mathbf{z}\sim q_\psi(\mathbf{z}|\mathbf{o})}\Big[\log p_\theta(\mathbf{o}\mid\mathbf{z})\Big]-\beta\,\mathrm{KL}\big(q_\psi(\mathbf{z}\mid\mathbf{o})\;||\;p(\mathbf{z})\big)\Big). \quad (3.18)$$

Increasing β encourages stronger regularization in the latent space and hence improves disentanglement, i. e. the latent space is more likely to have different dimensions corresponding to different and meaningful factors of variation from the data.

Due to the sequential nature of perception data, later models introduced a dynamics component into a VAE [67, 91, 106, 199]. This change enforces that the latent space encodes the right factors of variation needed to predict the future states or observations, which is a desirable property in a control system' state. In addition, imposing a structure on latent dynamics, such as linear transitions [91, 198], leads to state representations that simplify downstream control.

4

SUPERVISED STATE REPRESENTATION VIA ROBUST TRACKING

When a robot manipulator is in interaction with an object, the latter's appearance in a fixed camera tends to change due to several factors such as illumination, object pose, occlusion, and even motion blur. When using a template-matching approach to track the object, we typically have access to a single template image of that object. This template is either pre-selected by the system designer, or in scenarios where the robot is expected to interact with previously unseen objects, this template would correspond to a snapshot taken from the initial object detection.

As a consequence, the more an object's appearance is different from its initial appearance, the more challenging it is that such a tracker would correctly identify the 2D position of the object in the image. One could attempt to use, build, or train feature extractors that capture features that are invariant to the typical appearance changes experienced in such tasks. Hand-crafted features are typically tailored to particular tasks and do not transfer well to different contexts and environmental conditions. Recently, neural networks have been used as feature extractors. Siamese networks, in particular, are used to learn an embedding space for template matching [24]. At the time of this work, Siamese network-based methods were the most successful on most tracking benchmarks [101, 208]. However, regardless of the choice of features, it is possible that the initial template is not sufficient to extract features that would still be representative of the object after its appearance drastically changes.

This problem can be addressed by continuously updating the template to reflect the latest appearance of the object [71, 136, 191] or by constructing a representation of the template that implicitly handles this problem. As an example, Black and Jepson [27] used template matching in the eigenspace representation of the template, which is a compact way to encode a large collection of images with a few orthogonal basis images. A similar approach was also adopted in a different work [87]. Recently, Yang and Chan [211] proposed a method to construct dynamic memory networks that can adjust the template to the changing appearance of the target during tracking. This approach uses an long short-term memory network (LSTM) [75] as a memory controller to read and write templates to template memory based on memory neural networks [177, 202]. Reading in this context is selecting the right template, while writing is deciding whether to save an image as a template or not. Through this approach, their system creates a multi-template representation of the object. Lee, Choi, and Kim

[107] proposed a method that stores feature templates of the object and then creates a weighted combination of those templates to be later used for matching. Despite this, a mathematical combination of features may only be logical when we assume that all the stored templates are of the originally tracked object (and not a falsely tracked object due to drift).

What these methods all have in common is that they try to tackle the issue of change in appearance by altering the templates used for tracking. Similarly, in this chapter, we will introduce a method that builds template memory modules based on a neat mathematical property of the similarity metric used by Siamese trackers. Unlike the work in [211], the template memory module in this work is fully interpretable and is built based on an analytical mathematical framework, rather than using LSTMs. As a consequence, the method can be directly used on top of any Siamese-based tracker without any further training. Furthermore, extending a tracker with this framework barely affects speed. The primary reason behind these advantages is that we store templates that are distinct enough to represent the object and similar enough to the base template to prevent drifting to objects that are not relevant. To achieve this, the proposed framework relies on a measure of the diversity of the stored templates, as well as a lower bound on the similarity between the candidate and base templates.

4.1 BUILDING DIVERSE TEMPLATE MODULES

In a dynamic environment, an object can be subject to several condition changes, such as rotation (of the object or the camera), illumination, occlusions, motion blur and even changes in the shape of the object (e. g. due to a deformation). The main goal of this work is to present a framework, which enables building template modules accounting for all these problems and any other variations that the object could endure during tracking. The presented approach can be considered an extension to *any* template matching-based tracker which uses an inner product operation for similarity computation. The main idea is to find templates that are the farthest from each other in the feature space, as illustrated in Figure 4.1. By doing so, the framework builds a holistic representation of the object that is suitable for tracking. Hence, the method is called Tracking Holistic Object Representations (THOR).

The framework builds a memory composed of a **long-term module (LTM)** and a **short-term module (STM)**. Each of these modules stores templates that will later be used by the Siamese tracker. Each serves a different purpose. The LTM stores template images in a way that represents the tracked objects under diverse conditions (lighting, shape, etc.). Intuitively, if we assume a certain object has a fixed number of factors that affect its appearance in the image, this module aims to

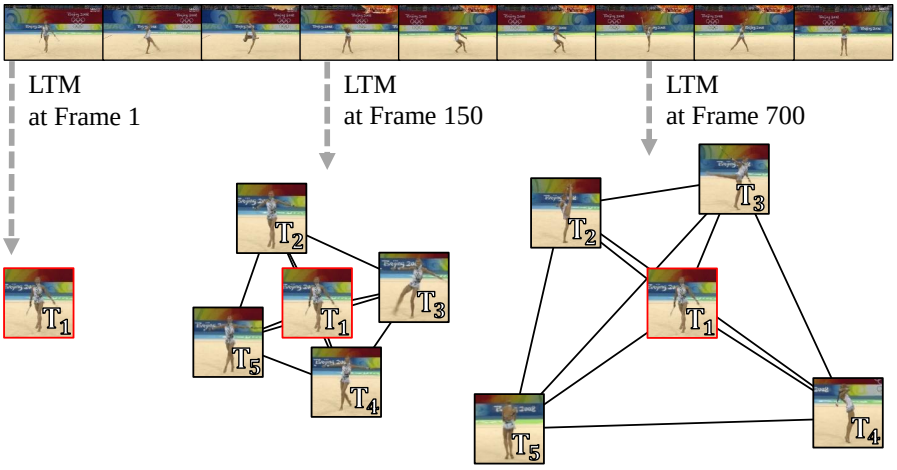


Figure 4.1: **Tracking Holistic Object Representations (THOR).** The task in the sequence *gym*, of the tracking benchmark OTB100 [208], is to track the gymnast. The goal of THOR is to maximize the diversity of the tracked object's representation while tracking. For explanatory purposes, we illustrate this representation, accumulated in the long-term module (LTM), in 2D. In reality, the representation occupies a high-dimensional space. Over time, the total volume of the parallelotope spanned by the templates increases. The base template T_1 always stays in the LTM and can be thought of as a fixed point of the parallelotope. Taken from [161].
A similar figure has previously been introduced in one of the author's previous publications [161].

store template images that sparsely represent different values of these factors. The LTM is used to track and re-detect the object in the long term. The STM stores templates that represent short-term variations of the object's appearance. The full system is shown in Figure 4.2. The idea of using long-term and short-term features for tracking has been exploited previously [78, 107]. However, THOR can be distinguished from previous methods based on the technique it uses to select the feature templates that are stored in memory. In the following, we explain the individual components of the architecture in detail.

4.1.1 *Long-term Module*

As mentioned previously, this module is responsible for finding and storing templates (in memory) that represent the diverse appearances that the tracked object could have. Therefore, its objective is to to store templates that maximize the diversity of information about the object. A naïve approach would be to store all tracked crops of an object. However, this approach has two main limitations. First, storing crops from all the tracking results quickly becomes infeasible memory-wise. This problem might be negligible when testing the algorithm on some fixed datasets. However, in real-world scenarios, such as robot manipulation, the length of a sequence is as long as the task execution.

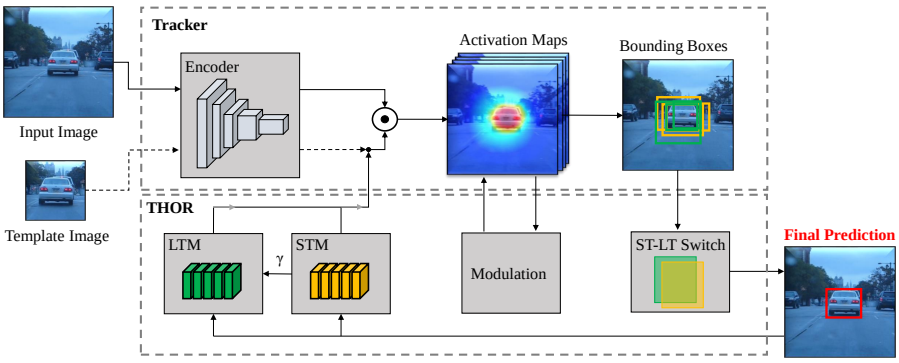


Figure 4.2: **System Overview.** The tracker and THOR can be considered separate components that exchange information. The *input image* and the initial *template image* are passed through an *encoder* (the template image only at the beginning of the sequence), transforming both into feature vectors in an inner product space. The activation maps are then computed with a dot product. For Siamese trackers, the encoder is a Siamese network and the dot product is a convolution. Over time, THOR accumulates long-term (LT) and short-term (ST) templates. Convolving the accumulated templates with the input image yields two sets of activations maps (corresponding to LT and ST templates). The *modulation* module calculates a weighted spatial average and multiplies it with all activation maps. Based on these activation maps, the tracker computes the bounding boxes. The box with the highest score in each set is fed into the *ST-LT switch* which determines which bounding box to use for the prediction. The final prediction is then fed back to the *STM* and *LTM* modules to decide whether to keep it or not. The STM also passes the diversity measure γ to the LTM.
This figure has previously been introduced in one of the author's previous publications [161].

Second, the batch size of the input to the Siamese trackers is limited by the GPU memory and number of cores. Hence, when dealing with long real-world sequences, the number of templates quickly explodes and a single pass with all templates becomes intractable. Consequently, it becomes feasible to maintain only a restricted set of templates, denoted as K_{lt}. As a result, a crop ought to be designated as a template solely if it provides supplementary object-specific tracking-relevant information beyond what has already been amassed by the existing templates.

In the real world, an object's state is affected by several object-specific properties such as material type, colour, shape and the dynamics of the object, but also by environmental properties such as illumination, temperature, etc. Only a certain amount of this information is recoverable from a 2D image of an object, which makes an exhaustive object description impossible. A practical alternative to this problem is to describe the object's state with visual features. Siamese trackers represent objects using visual features extracted from Siamese networks. Specifically, the template and target images are embedded into a feature space using the same network f. Given a target image T_i, the features are computed as follows: $\mathbf{z}_i = f(T_i)$. Template-matching approaches proceed by using the features of the template as a convolutional kernel. During tracking, the template kernel $\mathbf{z}_1$ is applied to the features of the input image to get the location of the highest similarity. Given two arbitrary templates T_j and T_k, the inner product of the corresponding feature vectors $\mathbf{z}_j \star \mathbf{z}_k$ is treated as a similarity measure of those two templates [24]. We notice that the space of n-dimensional visual features together with the convolution operator form an inner product space. Given a set of n feature vectors, we can construct a Gram matrix of the form

$$\mathbf{G}(\mathbf{z}_1, \cdots, \mathbf{z}_n) = \begin{bmatrix} \mathbf{z}_1 \star \mathbf{z}_1 & \mathbf{z}_1 \star \mathbf{z}_2 & \cdots & \mathbf{z}_1 \star \mathbf{z}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_n \star \mathbf{z}_1 & \mathbf{z}_n \star \mathbf{z}_2 & \cdots & \mathbf{z}_n \star \mathbf{z}_n \end{bmatrix}. \tag{4.1}$$

$\mathbf{G}$ is a square $n \times n$ matrix, where n is typically much smaller than the dimensionality of the feature space. Conveniently, the determinant of $\mathbf{G}$, called the Gram determinant, possesses a property that is useful for the LTM. Namely, this Gram determinant corresponds to the square volume of the n-dimensional parallelotope spanned by $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n$. Hence, the LTM uses this determinant as a measure of template diversity, when it selects which templates to store in memory. We can write the objective as follows,

$$\max_{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n} \Gamma(\mathbf{z}_1, \ldots, \mathbf{z}_n) \propto \max_{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n} |\mathbf{G}(\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n)|, \tag{4.2}$$

where $\Gamma(\mathbf{z}_1, \ldots, \mathbf{z}_n)$ is the volume of the parallelotope formed by the feature vectors $\mathbf{z}_i$ of the template T_i. The vectors $\mathbf{z}_i$ can be seen as the basis vectors of the feature space, which represent the manifold of the

tracked object in this embedded representation. One could also build a similar diversity measure by stacking all the features as columns in a matrix and computing the determinant of that matrix. However, the dimensions of such a matrix would grow with the size of the feature space — which tends in practice to be quite large — and the number of templates. This is in contrast to the size of $\mathbf{G}$ which is a square matrix with a dimension equal to the number of templates. The number of templates is a hyperparameter of THOR and hence can be set to feasible values. Given the objective in equation (4.2), THOR stores a template in the LTM if that new template increases the Gram determinant when replacing one of the templates already in memory. This means that we compute all the possible matrices with dimension n given the n stored templates and the one still under consideration. The templates corresponding to the highest Gram determinant would be in the LTM in the next step. The maximum number of templates in this framework is the dimensionality of the feature space D (ignoring memory restrictions). If $n > D$, the determinant would be zero. In reality, we set $n \ll D$.

So far, we have assumed that all the candidate templates do indeed correspond to the object. However, in practice, these candidates are the bounding boxes given by our tracker. Since the tracker is not perfect, it is highly likely that we will encounter candidate templates that correspond to completely different objects or perhaps just a part of the background. While being useless for tracking the target object, such templates would yield a high diversity in the template memory. As a consequence, they would be selected to be stored in the LTM. This could lead to the LTM becoming fully corrupted with false templates. To avoid this problem, we need a way to reliably detect such templates. One solution would be to set an upper bound on $|G|$. Such a value would be a hyperparameter of the method. However, tuning this value is not intuitive or straight-forward. Instead, THOR sets a lower bound on the similarity between the candidate template T_c and the original/base template T_1. The latter is the only ground truth available about the object. Hence, a template is only considered to be in the LTM if it satisfies $\mathbf{z}_c \star \mathbf{z}_1 > \ell \cdot \mathbf{G}_{11}$. ℓ is a hyperparameter that can be tuned for each task or dataset. It can be set differently to trade-off tracking performance against robustness to drift.

Setting such a static boundary would mean that the templates are restricted by a hypersphere in the feature space. In many cases, such a constraint would be too conservative. As it is unclear how to derive a proper lower bound in a way that ensures that we avoid drift while keeping diversity of object templates at a maximum, THOR is equipped with two various heuristics to handle this problem:

- *dynamic* lower bound: To account for the possible short-term changes in the object's appearance. This bound benefits from a

diversity measure of the templates in the short-term memory and subtracts it from the static bound

$$\mathbf{z}_c \star \mathbf{z}_1 > \ell \cdot \mathbf{G}_{11} - \gamma. \tag{4.3}$$

The faster the object's appearance is changing in the short-term — captured by γ being higher — the less conservative the bound should be.

- *ensemble* lower bound: This heuristic uses the same static bound, however requires it to be satisfied with respect to all the templates in the LTM instead of only looking at the base template. The condition becomes

$$\forall i \in \{1, \dots n\}; \quad \mathbf{z}_c \star \mathbf{z}_i > \ell \cdot \mathbf{G}_{ii}. \tag{4.4}$$

This heuristic enables much lower values for ℓ while still being robust against drift. This condition makes the LTM conservative at first but slowly allows for more diverse templates.

As per Bolme et al. [29], before computing the similarities between templates, the LTM multiplies the feature vector $\mathbf{z}_i$ with a tapered cosine window. This operation reduces the effect of the background of a template. This transformation alters the space in which the Gram determinant for the LTM is calculated. Nevertheless, the same mask is applied to all templates. Therefore, all LTM computations remain consistent in this new space. That is true regardless of the background-foreground ratio.

4.1.2 *Short-term Module*

When faced with abrupt movements or partial occlusion, the LTM is most likely to reject the candidate templates for being too dissimilar from its current template memory state. Hence, THOR relies on the STM to handle these scenarios. The STM has a fixed number of slots K_{st}. Without any precondition, the STM stores all detections in its memory in a first-in, first-out manner.

As mentioned in section 4.1.1, the long-term module expects a measure of short-term detections diversity. The STM calculates this measure γ using its own object templates. In fact, γ can be calculated using the Gram determinant as done in the LTM. However, the templates in the STM can be largely different from each other or even very similar in the early stages. Hence, the Gram determinant strongly fluctuates and as a consequence is hard to use by the LTM. In addition, since there are no bounds on the similarity of such templates, it is not possible to normalize this value. Instead, we calculate the diversity measure

$$\gamma = 1 - \frac{2}{N(N+1)\mathbf{G}_{st,\max}} \sum_{i<j}^{N} \mathbf{G}_{st,ij}. \tag{4.5}$$

In words, we sum up the upper triangle of the Gram matrix and normalize the sum by the maximum value in the Gram matrix. This puts γ in the range of $[0-1]$, the closer γ to 1, the more diverse the templates in the STM.

4.1.3 *Inference Strategy*‡

To get a predicted bounding box, we apply two methods during inference. *Modulation* aims to leverage all of the information that is contained in both STM and LTM. The *ST-LT Switch* determines which template yields the current best prediction and outputs the final bounding box.

At every frame, we get the activation maps of every template in both STM and LTM. To use the predictions of all templates, we compute a weighted spatial average over all activation maps. The weights correspond to the maximum scores for each template, i.e., if a template is more certain, it contributes more to the average. Every activation map is then multiplied by this average activation map and re-normalized.

By default, we always use the predicted bounding box of the STM, since it can handle short term challenges well. However, since no template stays in the STM permanently, it is prone to tracking drift. In visual tracking, drift is determined by calculating the intersection over union (IoU) of a predicted bounding box and the ground-truth [102]. We leverage this measure of drift and calculate the IoU between the two bounding boxes of the STM and LTM with highest scores. In this case, we treat the prediction of the LTM as ground truth since it is more robust against drift. If the $\mathrm{IoU}(\mathrm{STM}, \mathrm{LTM})$ is lower than a threshold $\mathrm{th}_{\mathrm{iou}}$, we use the prediction of the LTM and reinitialize the STM.

4.1.4 *Implementation Details*‡

To keep the memory updates and the forward pass efficient, we use two strategies: parallelization and dilation. For the memory updates, we need to compute the similarities between the template candidate and all templates in memory. The same also applies for a forward pass, where we need to compute the activation maps for all templates. The operation to compute the similarities is a 2D convolution, this means that we can calculate all similarities in parallel. Therefore, if the GPU memory is large enough, these operations slow down the tracker only slightly, see Section 4.2.2. Moreover, since consecutive frames are very similar in appearance, only every other frame is considered as a template. We set a constant dilation value of 10, i.e., every tenth frame is fed into the STM and LTM.

4.2 EXPERIMENTS‡

We build experiments to answer the following questions:

(Q1) Does the determinant increase throughout a tracking sequence and does it converge?

(Q2) What is the effect of THOR on the speed of the used trackers?

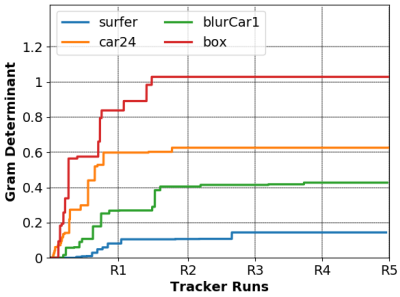
(Q3) Does THOR improve the performance of state-of-the-art trackers?

(Q4) What is the effect of each introduced concept (modulation, masking, lower bound, and short-term module) on the performance of the presented method?

Generally, THOR's underlying principle can be applied to any template matching tracker. We compare the following Siamese network-based trackers: SiamFC [24], SiamRPN [114], and SiamMask [197]. The tunable parameters of THOR are the number of memory slots in STM K_{st} and LTM K_{lt}, the IoU threshold of the ST-LT switch th_{iou}, the lower bound ℓ and α of the tapered cosine window. We use PyTorch for the implementation and the experiments were done on a PC with an Intel i9 and an Nvidia RTX 2080 GPU. In the following, we give a proof of concept, report the performance of on VOT2018 [102] and OTB2015 [208], and conduct an ablation study.

4.2.1 *Proof of Concept*

To validate that the Gram determinant of the feature templates truly represents the diversity of information about the object, we build the following experiment. We run a tracker (SiamRPN) together with THOR on sequences from OTB2015 and observe the normalized Gram determinant $|\mathbf{G}_{\text{norm}}|$ during the tracking. $\mathbf{G}$ is normalized against $\mathbf{G}_{11}$ to avoid numerical problems when calculating the determinant. At the end of a sequence, the final obtained templates are saved. We re-run the tracker while loading the previously saved templates and record the determinant again. We keep repeating this last step until the determinant converges. Surprisingly, the determinant does not stay constant after first reloading the templates. However, because the tracker yields different results with reloaded templates, the determinant keeps growing since it's exploring previously unseen candidate templates. Figure 4.3 illustrates this behavior. The convergence of the determinant represents the saturation of possible accumulated information from saved templates. Besides, we can observe that re-running the tracker with improved templates can also improve the AUC. The convergence of the determinant together with the improved AUC show that this measure truly enables the collection of good templates for tracking.



	R1		R5	
	$\lvert G_{norm} \rvert$	AUC	$\lvert G_{norm} \rvert$	AUC
surfer	0.0812	0.6169	0.1441	0.6562
car24	0.5983	0.8299	0.6258	0.8312
blurCar	0.2680	0.8125	0.4273	0.8141
box	0.8377	0.5837	1.0286	0.7542

Figure 4.3: **Proof of Concept.** Left: Convergence of the Gram determinant after repetitively re-running the tracker with THOR. Right: Gram determinant and area under curve (AUC) evaluated at the end of the first and last runs (R_1 and R_5) of the experiment.
This figure has previously been introduced in one of the author's previous publications [161].

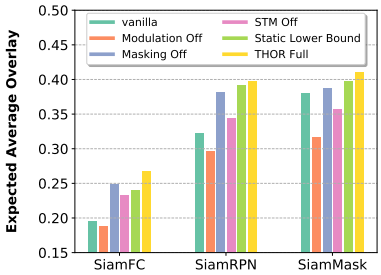
4.2.2 *State-of-the-Art Comparisons*

In this section, we determine the general performance regarding speed and the performance on the established visual tracking benchmarks VOT2018 and OTB2015.

Performance on VOT2018. On VOT2018, performance is measured in terms of accuracy, robustness, and expected average overlap (EAO), where EAO is used to rank trackers. Table 4.1 shows that THOR is able to improve all state-of-the-art trackers in terms of EAO. THOR-SiamRPN even pushes the performance back to current SotA results of trackers with much more sophisticated network architectures such as SiamRPN++ [112]. SiamRPN++ achieves an EAO of 0.414 while running at 35 FPS (on a NVIDIA Titan X). THOR-SiamRPN (dynamic) achieves an EAO of 0.416 while running at 112 FPS. The same strong improvements can be seen for robustness, which means that THOR mitigates tracking drift. Generally, the THOR-enhanced trackers perform slighly worse on accuracy. In some sequences, the tracker puts up with a loss in accuracy in order to keep the object tracked, by predicting a larger bounding box. A second reason is THOR's disposition to track the entirety of an object (which it does by design), therefore in sequence with e.g., face-tracking only, THOR can start to track the entire head, not only the face.

Performance on OTB2015. On OTB2015, performance is measured with the area under curve (AUC) and the mean distance precision. As shown in Table 4.1, THOR improves all trackers on both metrics. Especially precision is improved by adding THOR to the trackers. Generally, both dynamic and ensemble lower bound yield similar results.

Speed. Table 4.1 shows that THOR slows the trackers down, which is to be expected of a multi-template tracker since there are necessarily additional computations. The general bigger decline for SiamFC



	lower bound	
	dynamic	ensemble
Mean of $\lvert G_{norm} \rvert \Uparrow$	0.0261	**0.25164**
# drifted templates $\Downarrow$	**7**	16
# LT updates	599	1797
Relative drift $\Downarrow$	1.17 %	**0.89 %**

Figure 4.4: **Ablation Study.** Left: The effect on THOR's perfomance on VOT2018 when disabling modulation, masking, and the short-term module, or a static lower bound. Right: Comparison of the proposed strategies for the lower bound evaluated on OTB2015. *This figure has previously been introduced in one of the author's previous publications [161].*

can be explained with the expensive up-sizing operation in SiamFC, which is hard to parallelize in the given implementation. With a smaller and faster model, SiamRPN can reach speeds of 325 FPS, whereas THOR-SiamRPN can run at a respectable speed of 244 FPS. The experiments demonstrate that THOR still runs at a reasonable speed, especially in comparison to other multi-template matching approaches [107, 211]. It also shows that if a tracker is using a faster model, THOR can also be run at higher speeds.

Table 4.1: **Tracking benchmarks.** The attained performances of the trackers on VOT2018 and OTB2015. The main metric for ranking trackers is EAO (expected average overlay) on VOT2018, and AUC (area under curve) on OTB2015.
This table has previously been introduced in one of the author's previous publications [161].

		VOT2018				OTB2015		
Tracker	Lower Bound	Accuracy $\Uparrow$	Robustness $\Downarrow$	EAO $\Uparrow$	Speed (FPS) $\Uparrow$	AUC $\Uparrow$	Precision $\Uparrow$	Speed (FPS) $\Uparrow$
SiamFC	–	**0.5194**	0.6696	0.1955	**219**	0.5736	0.6962	**214**
THOR-SiamFC	dynamic	0.4977	0.4448	0.2562	99	**0.5990**	**0.7347**	97
THOR-SiamFC	ensemble	0.4846	**0.3746**	**0.2672**	69	0.5971	0.7291	80
SiamRPN	–	**0.5858**	0.3371	0.3223	**133**	0.6335	0.7674	**137**
THOR-SiamRPN	dynamic	0.5818	0.2341	**0.4160**	112	**0.6477**	**0.7906**	106
THOR-SiamRPN	ensemble	0.5563	**0.2248**	0.3971	105	0.6407	0.7867	110
SiamMask	–	**0.6096**	0.2810	0.3804	**95**	0.6204	0.7683	**97**
THOR-SiamMask	dynamic	0.5891	0.2388	0.3846	60	**0.6397**	0.7900	78
THOR-SiamMask	ensemble	0.5903	**0.2013**	**0.4104**	70	0.6319	**0.7929**	66

4.2.3 *Ablation study*

We introduced several concepts to enhance THOR's functionality. The LTM is improved by *template masking* and a *dynamic* or *ensemble lower bound*. The *STM* handles sequences with abrupt changes. We leverage the information of all templates through *modulation*. In the following, we conduct experiments to determine the influence of these concepts.

Tracking Performance. To measure the impact of the concepts on the tracking performance, we conduct an ablation study on VOT2018. We compare the performance of all trackers with THOR and without ("vanilla"). Then we disable one of the concepts to determine their influence on the final performance. Figure 4.4 (left) shows, that for all trackers the best performance (determined by EAO) can only be reached when all concepts are utilized. Turning off the modulation or the STM has the biggest negative impact on the final performance, which shows the importance of the respective concept.

Dynamic vs. Ensemble Lower Bound. To compare both proposed strategies for the lower bound, we record the normalized Gram determinant $|G_{norm}|$ at the end of every sequence in OTB2015. We then visually inspect the templates accumulated in the LTM and determine the number of drifted templates, i.e., when the tracked object is not in the center of the template. The relative drift is equal to the ratio of the number of drifted templates and the total number of updates. Figure 4.4 (right) shows that both strategies are effective in keeping the amount of drift low. However, the ensemble strategy manages to achieve a much higher mean of $|G_{norm}|$, indicating its ability to accumulate more diverse object representations (see also the qualitative comparison in Appendix A.1).

4.3 DISCUSSIONS

This work proposed a framework for building holistic visual object representations for tracking. The presented approach, THOR, can be used together with any other template-matching tracker that uses a similarity measure based on inner products. In summary, THOR builds a memory that accumulates short- and long-term object templates. The short-term templates are used to represent abrupt changes in object appearances, while the long-term templates are used to represent the diversity an object's appearance could have when the object is active. The main idea behind the method is to leverage certain detections during tracking as a way to accumulate more knowledge and pseudo-ground truth about the object. This comes in contrast to methods from the same family which only rely on the initial detection as the information about the object. To bypass memory and computational constraints, THOR selects this additional information based on a novel and efficient diversity measure. This diversity measure successfully enables THOR to collect useful object templates during tracking, as shown empirically in the experiments. Experiments further demonstrate that this framework can be used to augment multiple state-of-the-art trackers without the need for additional training or adaptation. This comes at an almost negligible speed cost. These two properties make this method very applicable to future works on Siamese tracking.

Although the presented framework demonstrates strong empirical performance, some aspects of it can still be improved. For instance, THOR's improvement over short sequences is not as great as for long ones. This is expected since short sequences do not allow for accumulating enough templates. A possible solution would be to make better use of the information from all the available templates rather than just using the prediction with the highest score. Such a strategy could also benefit long sequences. Furthermore, we only tested THOR on short-term tracking benchmarks. It would be interesting to observe how much improvement can be obtained for long-term tracking (for instance on OxUvA [190] or VOT-LT [102]). THOR could have even more impact on long-term tracking. However, there is a higher risk of drift in long-term tracking, which needs to be addressed by a properly chosen lower bound. Moreover, Siamese trackers are usually sensitive to the choice of hyperparameters which makes THOR similarly sensitive since it builds on top of them. This issue concerns the whole field of Siamese trackers and should be addressed. Finally, the current version of THOR does not require any additional training in addition to the training of the tracker itself. Hence, a possible future direction would be to train the tracker to also optimize the THOR objective.

In the grand scheme of things and following the overall topic of this thesis, the tracking results provide the robot learning pipeline with a state representation that captures information about the objects concerned in a given task. This information, together with the proprioceptive measurements from the robot, constitute the full-state information needed for many manipulation tasks. This combined information can then be used by any downstream task and motion planner to complete the given task. Alternatively, it can be used as an input to a reinforcement learning policy, which would then be trained to output actions that solve the given task. The choice of the RL algorithm should, in theory, not be affected by this state representation, making such an approach broadly applicable. Such a representation can surely be achieved with of-the-shelf object detection or tracking methods. However, at the time of this work, such methods were not robust enough to be used in an RL setup with requirements such as the Markov property that need to be satisfied. The author contributed to the integration of this tracker into multiple robotic manipulation demonstrators. Furthermore, the author contributed to an exhibition at a modern art museum (*Pinakothek der Moderne*) in Munich, Germany, where this work was showcased as part of a bigger robotics demonstrator.

5

EXPLORATION FOR STATE-REPRESENTATION LEARNING

Chapter 4 introduced a method for building state representations based on visual object tracking. This representation can then be used by a designed or learned planner to perform any given task. Although such a representation can be sufficient for many robot manipulation tasks, such an approach presents multiple disadvantages.

First, while it is possible to design states that are suitable for a single given task, the feasibility of designed states is questionable for the multi-task setup. Different tasks require different kinds of feedback. This is especially true when we think about the state of the robot's environment (excluding the robot state). When a robot is tasked with pushing objects on the table, the state of the environment can be characterized by the pose and velocity of the objects, as well as higher-order derivatives, depending on the requirements of the task. If we consider the task of sorting objects according to shape, the previous state needs to be augmented by additional shape-related information. In contrast, it is possible to learn state representations that can be leveraged by large families of tasks. This is possible since state representation learning methods typically attempt to recover the different factors of variation in the observation data (as discussed in chapter 3). Furthermore, by manually designing the state of the system, it is possible to unintentionally miss the information needed by the task planner to optimally infer successful task plans. For instance, in the pushing example, a system designer can choose to omit some nth-order derivative of the object's pose, assuming it is irrelevant to the pushing itself. However, the lack of this information could deny the planner the right information to estimate the effects of certain dynamics such as friction. This would in turn lead to suboptimal behaviors such as overpushing the object beyond its target. In contrast, if a state representation is learned together with the control, the state representation learning process can benefit from control signals to select the right information needed. In addition, designing the perception pipeline using a supervised learning method would in practice mean that the perception algorithm is trained using some already available (public) dataset. Therefore, the generalization of the learned perception components to the downstream task is not guaranteed. One could attempt to collect a dataset that contains data from the same environment as the downstream task. However, the labeling cost for such a dataset is very high and in many cases unfeasible. Hence, it is desirable to be able to learn the state representation mapping (perception) using the same data used for policy training.

Such data do not contain any state-related labels and usually contain sequences of raw observations, actions, and reward signals encountered when exploring the task. Of course, it is possible to learn the perception modules together with the policy using only reinforcement learning gradient signals [111]. Such an end-to-end approach is advantageous in the sense that the perception module also benefits from the reward signals and can hence pick the information needed to solve the downstream task. This type of process is typically very inefficient and requires millions of samples [17]. This in turn limits its applicability to real-world physical systems, where samples are expensive to acquire. Alternatively, we can use self-supervised learning methods to learn state representation mappings using the data from the same RL environment.

5.1 SELF-SUPERVISED STATE REPRESENTATION LEARNING

This integration of SRL methods in the RL pipeline can be done in multiple ways. Such an integration typically involves several design decisions. For example, the SRL component can be trained in an initial phase using data collected by a random policy. This phase would precede the classical RL training phase. Another approach is to alternate between the SRL training phase and the policy training phase using some user-set frequency. Alternatively, the SRL can be trained simultaneously with the policy and using the exact same data. It is important to note that this choice affects the diversity and quality of the data used to train the SRL. The more synchronized the SRL and policy updates, the less diverse the data. Bruin et al. [35] investigated the effectiveness of these different approaches on overall RL performance.

Another choice concerns the backpropagation of the policy gradients to the SRL modules. Although this detail is very simple to implement, it can have significant implications. First, if the policy gradients are indeed used to update the perception components, the latter's learned state representation would be biased to focus on state information that is relevant to the task the policy is trained on. As a consequence, the transferability of the learned representations to other tasks would be questionable. Laskin, Srinivas, and Abbeel [105] studied the impact of this choice on their method. Their experiments showed a clear advantage to letting the gradients flow back to the feature encoders. However, these experiments only considered the single-task scenario. Furthermore, it is unclear how the policy gradients can interfere with the SRL objective. For example, if a VAE is used for SRL, backpropagating the policy gradients through the encoder makes it uncertain whether the final state representation still adheres to the evidence lower bound of the original VAE objective. In contrast, if policy gradients are not used to update perception components, there is a risk that the learned state representation may not be

aligned with the goals of the policy and may not capture the relevant information for successful task execution.

Another important aspect is that the choice of using single-task or multitask data can have a significant impact. Single-task data refers to data collected specifically for the task at hand, while multi-task data refers to data collected from multiple tasks. Training with data from single tasks provides the advantage of being able to learn state representations that are suitable for the specific task. This allows for fine-tuning the state representation to capture the necessary information relevant to the task. However, this approach can be challenging and less feasible in a multi-task setup, where different tasks require different kinds of feedback. For example, the state representation needed for pushing objects on a table may differ from the state representation needed to sort objects according to shape. On the other hand, training with multitask data can leverage the power of state representation learning methods to capture the different factors of variation in observation data. By learning representations that can be used by a wide range of tasks, these methods have the potential to generalize well to new tasks. Additionally, by learning the state representation together with the control, the perception modules can benefit from control signals to select the relevant information needed for successful task execution.

Furthermore, the choice of the SRL method itself is important. Chapter 3 introduced multiple methods for self-supervised SRL. Each of these methods has its strengths and weaknesses, and the choice depends on the specific requirements of the task and the available data. For example, AEs can learn compact representations but may struggle with high-dimensional data, while generative models can capture the underlying distribution of the data but may be computationally expensive. Therefore, it is important to carefully select the SRL method that best suits the task at hand.

However, regardless of the SRL algorithm used, due to the relatively small size of the datasets used in this setup, the quality of the data samples plays a crucial role. Especially in methods that simultaneously learn the policy and SRL, the data distribution is typically very narrow. This is due to the repetitive and exploitative trial-and-error-based nature of RL training. The more successful the policy becomes, the less diverse the data samples contained in the trajectories collected by sampling this policy. As a direct consequence, the perception modules learned with such data can yield suboptimal state representations. Furthermore, due to the lack of diversity in the resulting datasets, the robustness and generalization of the learned SRL are questionable. In turn, reinforcement learning would suffer to consistently and robustly learn proper policies capable of performing the task. Methods that benefit from the policy gradient to update the perception modules can still recover from such suboptimal SRL but could require a substantially larger amount of interactions to achieve this.

Therefore, we notice the importance of having diverse datasets to train the perception components of agents. This chapter proposes an approach to training exploration agents that are only concerned with improving SRL. The aim is to build a method that can be used with a large array of SRL approaches, even future ones.

5.2 PERCEPTION-DRIVEN EXPLORATION

The majority of environments used for benchmarking deep RL algorithms provide dense reward functions. This means that the agent is almost always receiving some kind of reward signal to improve its policy. Once the reward is scarce and high-reward areas are harder to reach, these algorithms tend to struggle or even fail [36]. This argument is one of the main motivations behind research on exploration for deep RL. Popular paradigms for exploration include counts and pseudo-counts [18, 141], learning distributions over value functions or policies [140], and information gain based methods [79, 147, 164]. The main objective for all these methods is to improve the state-action distribution during training in a way that benefits RL. While this can positively affect the data used to train SRL, its effect is only indirect. Instead, the goal of this chapter is to provide a method that influences the agent-environment interaction in a way that directly benefits SRL. The goal here is to build an agent that is Curious about state Representation. Therefore, the proposed approach is referred to as CuRe for brevity. The emphasis in CuRe is on encouraging the exploration of states where the perception module struggles. Put simply, if the agent is bad at processing certain types of images, CuRe aims at increasing the change of reaching states where such images are observed. The hope here is that increasing the number of such images in the dataset would help the SRL module to learn how to process/understand them properly. For example, consider a robot arm that is tasked with picking up and manipulating objects. At some point during training, the perception module yields a state representation that only includes the positions of the objects in its environment. However, it struggles to accurately represent the orientation or shape of the objects, which can be crucial for successful manipulation. CuRe's role would then be to encourage the agent to explore and interact more frequently with objects that have varied orientations or unusual shapes. This increased exposure to challenging object configurations enhances the robot's perception and understanding of these properties through learning. As a result, the agent's SRL module can better encode and utilize this information, improving its grasp planning and manipulation capabilities.

The work presented in [167] is closely related to CuRe. The proposed method aims to optimize state entropy by employing random convolutional encoders. The technique incorporates a k-nearest-neighbor entropy estimator within the representation space, acting as an

inherent reward supplement for reinforcement learning. Analogous to CuRe, their approach eliminates the need for dynamic models during training. However, the utilization of a k-nearest neighbor entropy estimator could potentially lead to computational overhead if embeddings for all observations must be computed at each step. Alternatively, storing these embeddings in the replay buffer might impose significant memory requirements. Furthermore, a random encoder does not guarantee any notion of meaningful similarity between observations. In fact, in certain degenerate cases, the similarity in the representation space of a random encoder could be a measure of the dissimilarity of the states. Shelhamer et al. [169] proposed a method similar in principle to CuRe. Their approach also benefits from self-supervised losses to guide the SRL training. In the paper, the authors mention the possibility of using self-supervised losses as intrinsic rewards. Such an approach can be beneficial in the classical deep RL benchmarks such as Atari. However, for physical system control, learning a policy that is augmented by an intrinsic reward can be harmful to the stability and robustness of the final learned controller. CuRe proposes a simple remedy to this problem. Instead of augmenting the reward with an intrinsic one, CuRe leverages off-policy RL and learns an exploration-only policy in addition to the control policy used with the task reward only.

5.2.1 *Problem Formulation*

With a finite amount of data, it is not always possible to collect enough samples to learn a representation that is valid across the state subspace relevant to the task at hand. For instance, in environments with sparse rewards, the SRL training rarely encounters observations corresponding to high-reward regions and their surroundings. To improve the quality of the feature extraction and learned representations, it is important to encourage collecting data in states outside of the comfort zone of the SRL model.

Since SRL is trained with the data in the replay buffer D. The goal should be to have more data points in the replay buffer where the current perception module struggles. In other words, we want more samples where the SRL loss is maximized. The replay buffer contains multiple trajectories of the form $\tau = \{\mathbf{o}_{0:H}, \mathbf{a}_{0:H}\}$, where H is the horizon length. We omit the rewards here since they are irrelevant to the considered SRL methods. The joint distribution of the corresponding trajectory is the following:

$$p(\mathbf{o}_{0:H}, \mathbf{s}_{0:H}, \mathbf{a}_{0:H}) = p(\mathbf{s}_0) \prod_{t=0}^{H} [p(\mathbf{o}_t \mid \mathbf{s}_t) p(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1}) \pi_c(\mathbf{a}_{t-1} \mid \mathbf{s}_{t-1})], \tag{5.1}$$

where $\mathbf{s}_t$ is the underlying state, $p(\mathbf{s}_0)$ is the initial state distribution, $p(\mathbf{o}_t \mid \mathbf{s}_t)$ is the generative process of the observations, $p(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1})$ refer to the environment dynamics, and π_c is the policy used for collecting the data. The goal of CuRe can then loosely be written as

$$\max \mathbb{E}_{p_\tau}[\mathcal{L}_{SRL}(\tau)], \tag{5.2}$$

where p_τ is the distribution of the trajectories in the replay buffer and $\mathcal{L}_{SRL}(\tau)$ is the SRL error of the trajectory τ. The latter's distribution is according to equation (5.1). Therefore, it is evident that this loss function has the potential to influence four distinct distributions: the generative model, initial state distribution, dynamics, and policy. While some SRL methods — such as VAE — do have access to the generative model, optimizing this objective with respect to this model obviously defies the purpose of such models. Hence, SRL algorithms which have access to this model should keep it fixed in the optimization in equation (5.2). In fact, many SRL methods such as methods based on constrastive learning do not even consider data generation. As for the dynamics of the environment, they are typically imposed by the choice of embodiment and task. Hence, they cannot be changed. Which leaves the initial state distribution and data collection policy. The initial state distribution is typically fixed by the environment and is dependent on the environment resetting mechanism. Nonetheless, one could parameterize such a distribution with a neural network and optimize its parameters with this exploration objective. Although this may be an interesting avenue for future research, the current version of CuRe ignores this aspect.

Instead, in CuRe, the focus is on optimizing this objective with respect to π_c. Notice that the objective in equation (5.2) is very similar to the classical RL objective. Hence, we can simply train π_c with any RL algorithm if we compute the reward at each step by calculating the SRL error of the given observations. In practice, we could compute this reward directly at optimization time.

5.2.2 *Exploration during Task Policy Search*

The process described above can be easily used on its own to collect data to train the SRL. However, in many scenarios, it is desirable to simultaneously train the SRL and the actual RL task policy. Previous work on intrinsic motivation relied mainly on a weighted combination of extrinsic (task) and intrinsic rewards [147, 169]. While this can help improve exploration, the resulting policies would not be optimal for the task loss. Instead, CuRe uses two separate policies. One policy is trained using the task reward defined by the environment. Another policy is trained with the SRL loss as explained above. By doing so, we ensure that the task policy is purely optimizing the task reward, and additionally obtain a representation-curious agent capable of exploration for similar tasks in the same environment. More

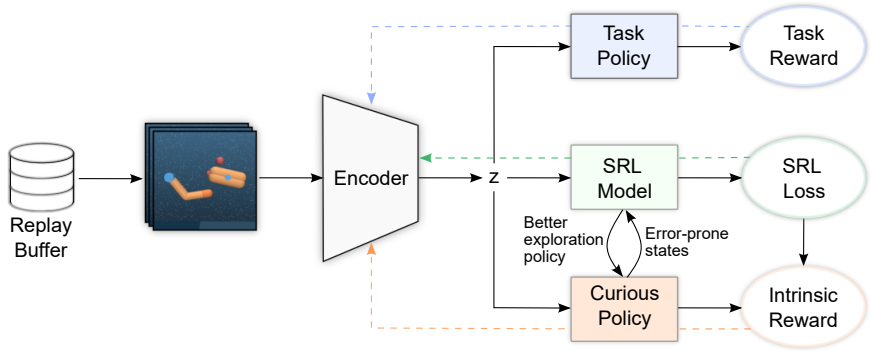


Figure 5.1: System Overview: our architecture is similar to the classical ones used for simultaneous state representation (SRL) and reinforcement learning (RL). Namely, an encoder is used to extract features from images and is trained together with an SRL model (e.g., decoder) to minimize the SRL loss. Simultaneously a task policy is trained to maximize the task reward, with the policy gradients flowing back to the encoder. In addition to the classical components, our method introduces a novel curious agent/policy, which is trained based on the SRL loss as an intrinsic reward. This creates an interplay between the SRL and the exhibited curious exploration behavior. The SRL guides the updates of the curiosity component, while the latter takes actions that lead to problematic and error-prone states. This in turn increases the diversity of observations.
This figure has previously been introduced in one of the author's previous publications [8, 9].

Algorithm 1

for each timestep $t = 1...T$ **do**
  $\epsilon \sim U(0,1)$
  if $\epsilon < p_c$ **then**
    $\mathbf{a}_t \sim \pi_c(.|\mathbf{o}_t)$
  else
    $\mathbf{a}_t \sim \pi_t(.|\mathbf{o}_t)$
  $\mathbf{o}_{t+1} \sim p(.|\mathbf{o}_t, \mathbf{a}_t)$
  $D \leftarrow D \cup (\mathbf{o}_t, \mathbf{a}_t, r_t(\mathbf{o}_t, \mathbf{a}_t), \mathbf{o}_{t+1})$
  $B \leftarrow SampleBatch(D)$
  $r_c \leftarrow UpdateSRL(B)$
  $UpdateTaskAC(B)$
  $UpdateCuriousAC(B, r_c)$

importantly, this choice allows our method to be used with both simultaneous and alternating approaches to SRL integration in RL. In addition, early experiments indicate that a separate curious policy leads to substantially higher reward areas, while the single policy approach could deteriorate the results in comparison to the baselines. Furthermore, adding the rewards together usually introduces extra hyperparameters to weigh the different terms [143]. It is important to note that having a separate policy is only possible when using off-policy RL algorithms such as soft-actor-critic (SAC) [64], which is why we use this method in this work.

Figure 5.1 illustrates the overall model and system in CuRe. A key strength of CuRe lies in its algorithmic flexibility, remaining independent of the chosen state representation learning algorithm. Our architecture encompasses not only the encoder and two policies but also an SRL model. The specific nature of this SRL model varies based on the adopted SRL methodology. For instance, in the case of an autoencoder-based approach, it functions as a decoder. Alternatively, it can embody a dynamics model, an identity transformation, or any computational element employed by representation learning methods to confine the latent space. It is important to note that the updates of both policies influence the encoder parameters ϕ, while only the SRL update affects the parameters of the SRL model θ.

During trial and error, if we only sample from the curious policy, the replay buffer would have great data for the training of the perception module. However, the policy training would suffer, since states with high SRL-error do not necessarily have high rewards. Sampling from task policy only would have the opposite effect. This problem embodies the classic exploration-exploitation dilemma encountered in reinforcement learning. It is slightly different than the classical problem, in the sense that exploration here is tailored towards perception and not control. Nonetheless, they are two problems with a very similar flavor. As a compromise, in CuRe, at each step, we draw actions from either the primary policy or the curious policy. The decision on which policy to employ hinges on the hyperparameter p_c, dictating the proportion of instances where exploratory actions are chosen. An alternative strategy involves drawing entire episodes via the curiosity policy. However, this method could produce vastly different samples, potentially unrelated to those encountered during the task, raising concerns about relevance. Moreover, a purely curious policy applied to complete episodes might compromise safety in real-world systems. Consequently, we restrict CuRe to exclusively sample steps (and not full episodes) via the curious policy, a design choice that mitigates these concerns.

More intuitively, after each training, the curious policy would sample actions that lead to states with a relatively high SRL error (under the current SRL model). Even a single action sampled from this policy would lead to a trajectory that is more likely to have states where

the perception module struggles. By sampling more often from it, the replay buffer gets filled with more such states. In the next update session, the SRL would encounter such states and, hopefully, learn how to handle them. If that is not the case, the SRL error for such states would remain high, and the policy would be rewarded again to visit them. Eventually, with repeated exposure to problematic states, the SRL/perception module should improve in representing them. By reducing perception errors, control and planning can only improve. This interaction between the curious policy and the SRL model/loss results in an interplay similar to the one observed in generative adversarial networks [58], as both modules are mutually beneficial to each other, and are trained in an adversarial setting. This interplay is illustrated in Figure 5.1. The overall approach is summarized in algorithm 1.

5.3 EXPERIMENTS‡

We design experiments to answer the following questions:

(Q1) Can we train a curious policy to increase the visitation of high SRL error states?

(Q2) How does CuRe affect the performance, sample efficiency, and training stability of vision-based RL methods? Can CuRe be successfully integrated with multiple SRL methods?

(Q3) Does CuRe-driven SRL pretraining improve the performance of vision-based RL on downstream tasks?

5.3.1 *Setup & Baselines*

To answer these questions, we experimentally evaluate our method on 6 continuous control tasks from the Deepmind Control Suite [186]. The chosen tasks aim to cover a wide range of common RL challenges, such as contact dynamics and sparse rewards. The tasks we use are `reacher_easy`, `cartpole_swingup`, `ball_in_cup`, `finger_spin`, `finger_turn`, and `reacher_hard`. As deep learning models could be energy inefficient [176], we use only subsets of these tasks for minor experiments that are only aimed at validating simple aspects of our method.

The main goal of our experiments is to validate the effectiveness of CuRe on improving the performance of already existing SRL-based approaches to vision-based RL. To do so we use two such algorithms as baselines and compare their performance with and without CuRe. To validate, that the method is agnostic to the choice of SRL algorithms, we experiment with two different methods. Namely, we use a combination of SAC with RAEs as in *sac_ae* [212] and a combination of SAC with contrastive learning based on *curl* [105]. We chose those two SRL

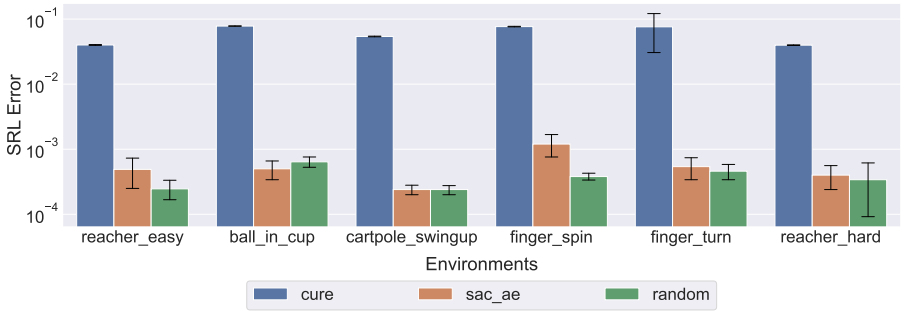


Figure 5.2: State representation learning (SRL) error encountered in trajectories sampled with three different policies: *random*, *sac_ae*, and our curious policy (*cure*). The bars represent the mean error per step. The error bars represent the minimum and maximum encountered errors. Our method leads to the visitation of high SRL error states, around two orders of magnitudes more than the random and task policies (*sac_ae*).
This figure has previously been introduced in one of the author's previous publications [9].

methods since their integration in RL is fairly recent while also being well-established in robotics applications. We refrain from comparing our approach to classical exploration methods since the two have different goals: classical exploration in RL is concerned with improving the sample diversity for RL while our method is aimed at encouraging the visitation of SRL-problematic states (discomfort zones). Hence comparing methods from these two categories could be misleading. Both baselines and our method are implemented using PyTorch [146]. For simplicity, we use the same hyperparameters for all experiments except for the action repeat value which changes per task, according to [68]. The actor and critic networks for the RL agent and the curious agent are trained using the Adam optimizer [97], using default parameters. For implementing SAC, we follow the training procedure detailed in [212].

5.3.2 *Results*

In the next three sections, we attempt to empirically answer Q1, Q2, and Q3, respectively. For each question, we design a fitting experiment and examine the data for corresponding answers.

5.3.2.1 *Visiting High SRL Error Regions*

Figure 5.2 shows the SRL error encountered when sampling actions from three different policies. The first policy generates *random* actions within the action space of the environment. The second one is trained with *sac_ae*, and the last one is a CuRe-based curious policy that maximizes the SRL error without a task reward. While *random* and *sac_ae*

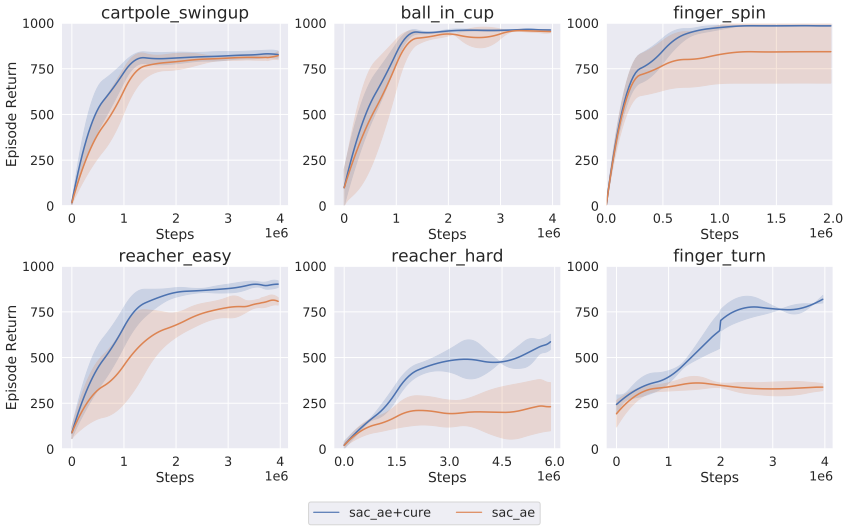


Figure 5.3: Training curves on six continuous control tasks from the Deepmind Control Suite [186]. The plots show the mean episode rewards of two algorithms. The first one is a baseline (*sac_ae*). The second method combines the same baseline with CuRe (*sac_ae+cure*). In all environments, our method exceeds the performance of the baseline. For easier tasks, the curious exploration either stabilizes the training or improves the maximum achieved reward. For the more difficult tasks, such as `finger_spin`, `finger_turn`, and `reacher_hard`, the additional curiosity objective allows to improve the average reward, where the baseline fails to reach high-reward areas.
This figure has previously been introduced in one of the author's previous publications [9].

have similar mean errors per step, our method leads to the visitation of states which have on average an SRL error that is around two orders of magnitude higher. This confirms that CuRe fulfills its goal of increasing the probability of visiting high SRL error states.

5.3.2.2 *CuRe-based Exploration During RL*

To answer (Q2), we study the effect of integrating CuRe into two different baselines, namely *sac_ae* and *curl*. The integration is based on algorithm 1. Figure 5.3 shows the task reward for *sac_ae* with and without CuRe. In all environments, our method exceeds the performance of the baseline. Specifically, for tasks where the baseline does not show any signs of improvement, such as `reacher_hard` and `finger_turn`, CuRe leads to exploring high-reward areas, as can be seen when looking at the maximum rewards achieved in those environments. For simpler tasks such as `reacher_easy` and `finger_spin`, our method approaches the maximum environment rewards, while *sac_ae* converges to 80%. In addition, CuRe stabilizes the training and seems to reduce the reward variance significantly. This last feature is not given enough attention in RL research. However, in real-world scenarios, when deploying RL agents, there could be cases where only one training run is possible. An algorithm with lower reward variance could guarantee a sufficiently good policy, while it's hard to say the same when this condition fails. This effect can also be seen for `cartpole_swingup` and `ball_in_cup`. We observe that CuRe has a minor effect on the maximum reached reward for these last two environments. This could be attributed to the already good performance of the baseline on these tasks. In fact, in these environments, *sac_ae* already approaches the performance achieved by SAC trained with the true states [212]. Nonetheless, the additional curious exploration objective accelerates the convergence of all evaluation tasks, thus improving the sample efficiency, which is one key limitation of state-of-the-art model-free algorithms. Our experiments show that CuRe becomes more effective when the task complexity increases.

To study the effect of CuRe on *curl* [105], we run experiments on the four environments where CuRe had the most influence on *sac_ae*. Figure 5.4 shows the reward plots for *curl* with and without CuRe. Similar to our previous results, CuRe has a positive impact on the overall performance, sample efficiency, reward variance and stability of training. This improvement is not as big as the one observed in our *sac_ae* experiments. However, this difference is understandable, since *curl* is a more recent algorithm and has previously shown better results on similar deepmind control suite tasks [105]. Despite that, when looking at results on `finger_turn` (Figures 5.3 and 5.4), CuRe applied to *sac_ae* reaches a higher final episode reward than vanilla *curl*. Additionally, we observe that *sac_ae+cure* has a better sample efficiency than *curl* in the `finger_spin` environment.

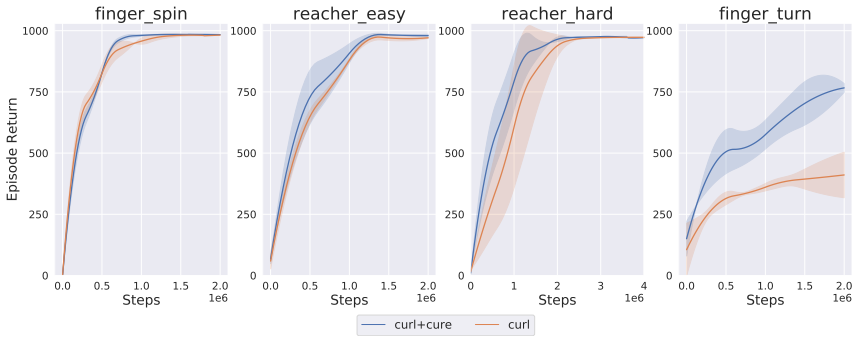


Figure 5.4: Training curves on four continuous control tasks from the Deepmind Control Suite [186]. The plots show the episode rewards of two algorithms. The first one is a baseline (*curl*). The second method combines the same baseline with our curious policy (*curl+cure*). In all environments, our method improves the overall performance, sample efficiency, or reward variance.
This figure has previously been introduced in one of the author's previous publications [9].

5.3.2.3 *Effect of Pretraining*

In addition to our main results, to assess the quality of the learned representation with CuRe, and to answer (Q3), we study the effect of two different pretraining procedures on *sac_ae*. Namely, we look at pretraining the RAE using samples collected either using a random policy (random-pretraining) or using a policy trained with CuRe only, without any task reward (CuRe-pretraining). For both options, we perform the pretraining for half a million steps. We also compare the performance of those two variants to the case where no pretraining is performed at all (vanilla). The results are shown in Table 5.1. For all six environments, the best results are obtained when using one of the two pretraining mechanisms. In most cases, CuRe-based pretraining leads to better performance than random-pretraining. This become especially apparent for tasks where the vanilla method struggles, such as `reacher_hard` and `finger_turn`. However, for the `ball_in_cup` environment, CuRe-pretraining seems to deteriorate the performance when compared to both vanilla and random-pretraining. This could be attributed to the simplicity of the task, which reduces the need for SRL and SRL-tailored exploration. In general, although CuRe is beneficial for both SRL pretraining (Table 5.1) and RL (Figure 5.3), we observe that it is more effective during task learning than in the pretraining phase.

5.4 DISCUSSIONS

In vision-based reinforcement learning, SRL plays an important role in learning. This is valid whether SRL is learned only through task reward or also using supervised or self-supervised learning methods.

Table 5.1: Comparison of performance (in terms of episode reward) of different versions of *sac_ae*: vanilla is the original algorithm [212], random-pretraining and CuRe-pretraining refer to the cases where the vanilla procedure is preceded by an RAE pretraining phase using data collected with a random policy and a CuRe-based policy respectively.
This table has previously been introduced in one of the author's previous publications [9].

methods	cartpole_swingup	ball_in_cup	finger_spin	reacher_easy	reacher_hard	finger_turn
vanilla	833± 27	953 ± 4	820 ± 144	714 ± 113	169 ± 179	229 ± 135
random-pretraining	784 ± 12	**955 ± 10**	975 ± 3	615 ± 129	84 ± 33	256 ± 40
CuRe-pretraining	**846 ± 25**	504 ± 187	**981 ± 7**	**804 ± 52**	**431 ± 40**	**402 ± 58**

In the latter case, the data used for training the perception models are typically collected using some policy trained on the downstream task. This results in narrow data distributions, which make the learned perception modules less robust and generalizable for real-world requirements. Hence, this work examined perception-centric exploration in vision-based reinforcement learning. This chapter introduced CuRe, a method for learning policies that are curious about state representation. These policies aim to take actions that would lead to states outside of the comfort zone of the current perception module. CuRe learns this curious policy separately from the task policy, and benefits from the ability of off-policy RL algorithms to be trained with data sampled by a different policy. As a result, the curious policy can be used to pretrain the SRL component or can also be used during the policy search to improve the sample diversity in the replay buffer in a way that benefits perception. Experiments with CuRe showed an improvement in the visitation rate of the SRL-problematic states, and as a result, an improvement in the RL performance and sample efficiency. One major advantage of this method is its applicability to a wide array of SRL methods without any constraints, besides access to the SRL loss. In addition, the implementation of this method is very simple and requires minor modifications to preexisting RL pipelines.

5.4.1 *Limitations*

CuRe introduces the hyperparameter p_c, which determines the sampling rate from the exploration policy. In other words, this hyperparameter controls the exploration-exploitation dilemma as conveyed throughout this chapter. Although tuning this hyperparameter is not difficult, it does play an important role in the performance of CuRe. However, this is a common problem in RL. Nevertheless, a more natural integration of the curious policy sampling or a way for tuning this hyperparameter remain needed for the algorithm to be more widely used. Furthermore, despite being theoretically applicable to any SRL

method, the effectiveness of CuRe on methods beyond the ones studied in this work remains open.

5.4.2 *Outlook*

The current version of CuRe only looked at visual perception. In theory, the method should also be applicable to other perception modalities or even a multimodal application. The latter presents an opportunity to test whether this perception-driven exploration could facilitate sensor fusion by the means of seeking states for which the combined multimodal observations are hard to process. The reason why we picked visual perception for this first work is due to its importance for robotic manipulation, as well as the difficulty of building and learning visual perception modules for robotics. Another interesting modality would be tactile perception. In most manipulation tasks, high-reward regions are encountered when the robot is in contact with or in the vicinity of objects. Using a method like CuRe for exploration could incentivize reaching and interacting with objects since the effect of contacts would not be immediately captured by the SRL module. Hence, such states would be labeled as problematic and sought out by CuRe's exploration policy.

Another interesting avenue for future work is transfer learning. Perception-driven exploration can play a role in learning perception components that can be positively transferred to other tasks. In contrast, typical RL exploration that is not tailored towards perception does improve the state exploration of the system but does not necessarily optimize for having a more robust visual system. Studying the effects of transferring CuRe-based learned SRL/perception modules has great potential.

Similarly, multi-task RL can benefit from this paradigm. In fact, one of the components that can be easily shared across tasks is perception. Different tasks may require different task and motion planner as well as low-level controllers. Perception, on the other hand, if it yields a task-agnostic state representation, can easily be shared across multiple tasks. This is at least true for environments that share the same embodiment and some task similarities. In a multi-task scenario, CuRe can be used for collecting data by sampling actions across multiple tasks. The perception modules can then be trained using the resulting mixed dataset.

Furthermore, it would be interesting to look at information-theoretic formulations of perception-driven exploration. Such a formulation could take advantage of well-established principles from information theory such as entropy maximization, information gain, or empowerment.

Finally, real-world robot learning is potentially the most interesting future avenue of extensions for this work. While access to interactions is cheap in simulation, real-world experiments are very

expensive. Any improvement to sample efficiency can bring robot learning a step closer to being more feasible. In addition, real-world sensors are a lot more noisy, and their measurements are typically harder to capture by SRL methods compared to simulated observations. Perception-driven exploration and the promise of more robust perception modules become more valuable in such a setting. However, like any exploration work, applying such an approach in the real world comes with safety concerns. It is unclear what those exploration actions could look like on a real system. This itself might be the motivation for another line of research combining these methods with robot safety measures or safe exploration techniques.

Part III

ROBOT ACTION REPRESENTATIONS

This part of the thesis is based on ideas, theories, and experiments that appeared in the following publications and under-review papers:

Elie Aljalbout, Ji Chen, Konstantin Ritt, Maximilian Ulmer, and Sami Haddadin. "Learning vision-based reactive policies for obstacle avoidance." In: *Proceedings of the 2020 Conference on Robot Learning*. Vol. 155. Proceedings of Machine Learning Research. PMLR, 2021.

Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. "On the role of the action space in robot manipulation learning and sim-to-real transfer." In: *IEEE Robotics and Automation Letters* (2024).

Elie Aljalbout, Maximilian Karl, and Patrick van der Smagt. "CLAS: Coordinating multi-robot manipulation with central latent action spaces." In: *Proceedings of The 5th Annual Learning for Dynamics and Control Conference*. 2023.

Direct quotes from the papers are highlighted in gray font color. Sections or subsections with titles marked with ‡ correspond to (sub-)sections from the author's corresponding previously published papers.

Supplementary material related to this part's chapters is collected in Appendix B.

6

THE ROLE OF THE ACTION SPACE

Despite RL being primarily focused on control, recent deep RL literature has shown an overwhelming larger interest in the state and observation space aspect of the problem. For benchmark problems such as games, the action space is predefined by the task and environment and alternatives often do not exist. In robotics, particularly in robotic manipulation, multiple options are possible. For instance, the control of a robot manipulator can either be framed in the task or configuration space. Task space control defines the problem with respect to a frame attached to a rigid body part of the manipulator such as the end-effector of the robot. Control is then typically defined in a Cartesian space relative to this frame. Hence, it offers an intuitive and interpretable interface for humans and an easy integration with task objectives which often involve Cartesian-space variables or goals. On the other hand, configuration space control directly dictates the motion of the robot's joints. Such an interface is less intuitive to humans but more native to the robot's embodiment. In addition, configuration space control is more readily suitable for defining control constraints related to the robot's motion and interaction behaviors. Besides the task/configuration distinction, a manipulator's controller can be defined with respect to different orders of derivative of positions. For instance, one could define a position, velocity, or acceleration control law, or even go beyond pure motion control via force or impedance control. The choice of controller is central to the success of a task. For instance, motion control is simple and suitable for reaching-like tasks, while force control can be more suitable for tasks involving interaction with the environment.

Similarly, in RL, the choice of the action space completely alters the problem's definition conceptually, theoretically, and in practice. Changing the action space would result in a different MDP and hence different complexity, sample requirement, exploration behavior, and even system capability. Complexity-wise, a good action space could abstract challenging control aspects that the policy would otherwise need to handle. For instance, using a position-control action space can abstract the joint torque computation aspect from the policy. This would effectively reduce the policy's complexity, as well as that of the environment dynamics from the perspective of this high-level action space. As a result, an RL agent exploring an environment equipped with such an action space could more easily focus on the task goals without having to understand the connection between torques applied to its joints and the resulting motion. Hence, such an abstraction could be beneficial for exploration and the resulting sample re-

quirement. However, abstraction could also limit the policy's flexibility and control over the robot. For instance, a position-based policy could only indirectly apply a force on the environment, while the less abstract torque or force action spaces could natively enable that. In addition, when attempting to transfer policies trained in simulation to the real world, the choice of action space could play a role in the sim-to-real gap and help or impede a positive transfer.

These aspects of action space design create a trade-off between high-abstraction spaces which reduce complexity and data requirements, and low-level spaces which give the policy full control, yet make it harder to learn in low-data regimes. It becomes clear that the choice of action space is central to the robot learning problem, and that the field requires novel methods that are more suitable for learning-based control. An ideal action space should strike the right balance between abstracting irrelevant control aspects from the policy and giving it the right amount of control to achieve the task. One motivation for learning-based robot control is to be able to learn a large set of tasks with one or a limited set of algorithms, with minimal human supervision. This brings forward the question of whether we could design action spaces that are suitable for learning the largest possible set of robotic tasks. This chapter aims to provide a better understanding of the action space in manipulation. It also aims to prove that this choice is of high importance to the field, and to identify characteristics that are favorable in the design of such methods.

In the literature, recent work has explored the choice of action space in manipulation [10, 126], flying [92], and locomotion [51, 149] policies. The corresponding findings sparked the development of novel action spaces suitable for different families of tasks. For instance, for manipulation, the force exerted by the robot on its environment plays a crucial role. Multiple proposed action spaces for manipulation allow the policy to control this aspect, either by some direct means of force application [19, 89, 189] or implicitly via impedance control [28, 121, 126]. Furthermore, several methods explored the use of movement primitives in the action space [4, 15]. Another important choice is between configuration or task-space control. Robot learning literature includes methods with both configuration [4, 81, 173, 209] and task space [51, 121, 126] action spaces. Ganapathi et al. [56] propose an approach to implicitly combine these two modes using a differentiable forward kinematics model. Interestingly, it seems like the majority of sim-to-real works use some kind of configuration space action spaces [3, 69, 81, 184, 209]. The reason behind this is yet unclear. More recently, several approaches have been proposed for learning latent action spaces that can either help reduce the dimensionality of the policy's control to the task manifold [11, 219] or serve a different purpose such as coordinating multi-robot tasks [7].

To better understand this aspect of robot learning, there have been multiple studies concerning the action space in robotics [92, 149, 193].

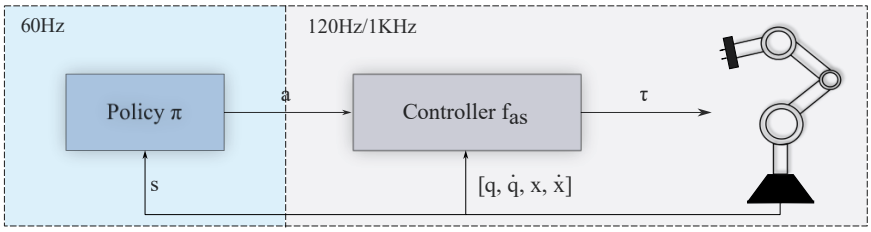


Figure 6.1: The policy outputs an action **a**, which is then transformed into joint torques τ using a select controller f_{as}. The policy and controller receive feedback from the environment. Each action space is defined by the choice of the controller and the way the action is treated in the controller. The policy runs at a 60Hz frequency and the controller runs at a frequency of 120Hz and 1KHz in the simulation and the real world respectively.
This figure has previously been introduced in one of the author's previous publications [5].

These studies span robotic applications including locomotion [149], manipulation [193], and flying robots [92]. However, the majority of these studies focus on a very limited set of action spaces or tasks. Multiple of them, study this problem in simulation only [149, 193]. In contrast, Kaufmann, Bauersfeld, and Scaramuzza [92] study the transfer of flying policies from simulation to real under three different types of action spaces. This chapter presents a study on the role of the action space in exploration, learning capability, emerging properties, and sim-to-real transfer. The study includes 13 different action spaces and 2 different manipulation tasks.

6.1 ACTION SPACES FOR MANIPULATION‡

For arm manipulation, one of the most native RL action spaces is one that expects **joint torque (JT)** commands. This corresponds to $\boldsymbol{\tau} = \mathbf{a}$, where $\boldsymbol{\tau}$ is the vector of joint torques. Such an action space gives the policy full control over the robot. When given the right observations, such a policy can internally learn to control the motion and forces exerted by the robot. This action space was popular in early works using deep RL [116, 196]. However, learning a policy with this action space can be very complicated since the policy would need to either understand or implicitly handle both the kinematics and dynamics of the robot to fulfill the task. This is due to the fact that the reward is typically expressed using task space properties.

Alternatively, a low-level controller $f_{as} : \mathcal{A} \longmapsto T$ can be integrated in the action space to convert higher-level policy actions into the space of joint torques T of the robot. This concept is illustrated in Figure 6.1.

6.1.1 *Configuration action spaces*

Configuration action spaces consist of all action spaces that expect a configuration-space action from the policy. At the center of all these action spaces is the same controller, namely a joint impedance controller (JIC). JIC regulates the behavior of a robot manipulator's joints and allows specifying desired stiffness, damping, and inertia characteristics for each joint. This allows the robot to be compliant with its environment. The JIC control law is

$$\boldsymbol{\tau} = f_{\text{JIC}}(\mathbf{a}) \tag{6.1}$$

$$\boldsymbol{\tau} = \mathbf{K}(\mathbf{q}_d - \mathbf{q}) + \mathbf{D}(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}), \tag{6.2}$$

where $\boldsymbol{\tau}$ denotes the commanded joint torque, $\mathbf{q}_d$ and $\dot{\mathbf{q}}_d$ denote the desired joint positions and velocities respectively, and $\mathbf{q}$ and $\dot{\mathbf{q}}$ denote the actual joint positions and velocities of the robot given as feedback. $\mathbf{K}$ and $\mathbf{D}$ are the stiffness and damping matrices. Note that we omit the gravity vector from our controller equations for clarity and simplicity. In practice, we use isotropic gains, i.e., these matrices are diagonal. Given this control law, we can define two different base configuration action spaces:

- **Joint Velocities (JV)**: sets $f_{\text{JV}} \leftarrow f_{\text{JIC}}$, and $\dot{\mathbf{q}}_d = s(\mathbf{a})$ and $\mathbf{q}_d$ is a first-order integration of $\dot{\mathbf{q}}_d$;
- **Joint Position (JP)**: sets $f_{\text{JP}} \leftarrow f_{\text{JIC}}$, and $\mathbf{q}_d = s(\mathbf{a})$ and $\dot{\mathbf{q}}_d$ is a first-order differentiation of $\mathbf{q}_d$.

$s(\mathbf{a})$ is a function that scales the action vector to the limits of the corresponding output vector. The differentiation and integration steps are often ignored in RL action space implementations, despite being present in the most common robot control libraries. Not including these steps in the simulation would create an additional sim-to-real gap.

6.1.2 *Task action spaces*

Task action spaces are defined using variables that are in the task/-Cartesian space of the robot. In the absence of accurate system identification, we can use the following Cartesian impedance controller,

$$\boldsymbol{\tau} = \mathbf{J}(\mathbf{q})^{\mathsf{T}}\big(\mathbf{K}(\mathbf{x}_d - \mathbf{x}) + \mathbf{D}(\dot{\mathbf{x}}_d - \dot{\mathbf{x}})\big), \tag{6.3}$$

where $\mathbf{J}(\mathbf{q})$ is the Jacobian matrix for the current robot configuration $\mathbf{q}$, relating joint velocities to Cartesian velocities, $\mathbf{x}_d$ and $\dot{\mathbf{x}}_d$ are the desired Cartesian poses and velocities, $\mathbf{x}$ and $\dot{\mathbf{x}}$ are the current Cartesian poses and velocities respectively. However, this formulation can be tricky to use when the $\mathbf{x}_d$ and $\dot{\mathbf{x}}_d$ are generated by an RL policy. This

is due to the non-smooth nature of RL action trajectories. Of course, this problem can be handled by introducing interpolators or cubic spline fitting. But such solutions involve multiple design choices and hyperparameters, meaning that an ideal solution is task-specific. Instead, we transform the Cartesian actions into joint velocities, and then use a joint impedance controller. We found this approach to be very good at handling the non-smooth policy actions, without introducing any additional sim-to-real gap. We have two base task action spaces:

- **Cartesian Velocities (CV)**: sets $\dot{\mathbf{x}}_d \leftarrow s(\mathbf{a})$, transforms $\dot{\mathbf{x}}_d$ into $\dot{\mathbf{q}}_d$ using inverse kinematics (IK), and then uses the f_{JIC} in the same fashion as in the joint velocities action space;
- **Cartesian Position (CP)**: sets $\mathbf{x}_d \leftarrow s(\mathbf{a})$, and then uses a proportional control to obtain $\dot{\mathbf{x}}_d$ from $\mathbf{x}_d$. This step naturally results in smooth $\dot{\mathbf{x}}_d$. Given $\dot{\mathbf{x}}_d$, this action space proceeds as in the CV action space.

We use the pseudoinverse IK method with a null-space controller that pushes the joints toward their default positions.

6.1.3 *Delta action spaces*

Delta action spaces are based on the base action spaces defined previously. In contrast to the base action spaces, delta action spaces set the control targets $\mathbf{v}_d$ relative to the current system feedback or to the control targets of the previous policy step. This distinction creates two classes of delta action spaces:

- **One-step Integrator (OIΔ)**: uses the robot feedback $\mathbf{v}$ to set $\mathbf{v}_d \leftarrow \mathbf{v} + c \cdot \mathbf{a} \cdot dt$;
- **Multi-step Integrator (MIΔ)**: recurrently sets $\mathbf{v}_d \leftarrow \mathbf{v}_d + c \cdot \mathbf{a} \cdot dt$.

Depending on the choice of base action space, $\mathbf{v}_d$ is a control target vector, and can correspond to $\mathbf{q}_d$, $\dot{\mathbf{q}}_d$, $\mathbf{x}_d$, or $\dot{\mathbf{x}}_d$. Similarly, $\mathbf{v}$ is a control feedback vector, and can correspond to $\mathbf{q}$, $\dot{\mathbf{q}}$, $\mathbf{x}$, or $\dot{\mathbf{x}}$. dt is the step duration and c is a positive constant hyperparameter. Instead of the scaling performed in non-delta base action spaces, we clip the target $\mathbf{v}_d$ to the limits of the corresponding output space after updating it. This means that each base (configuration or task) action space has two additional variants, resulting in 12 action spaces, not including the joint torques action space. Despite relying on the base action spaces, the delta variants have their unique properties. For instance, due to the relative changes in the control targets, the magnitude of the target change in one step is bound by $c \cdot dt$ given that $a \in [-1, 1]$. This property can be helpful in imposing smoothness constraints on control target trajectories, even if the policy output is unconstrained. Additionally, if we set c to be the positive bound of the derivative of

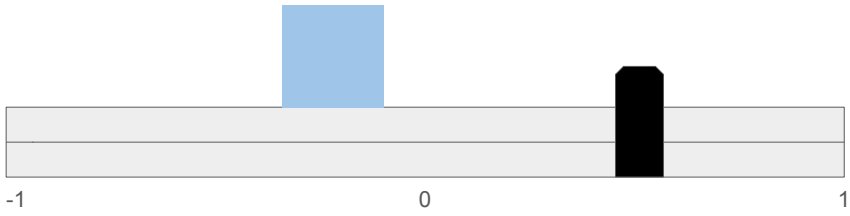


Figure 6.2: A simple 1D task with a single prismatic joint and an object. The task is to move the joint to push the object into a target position.

the corresponding control feedback variable $\mathbf{v}$, each delta action space would approximately be equivalent to an action space of a higher-order derivative than its base action space. For instance, a delta joint velocity action space would be approximately similar to a joint acceleration action space. However, when sampling actions $\mathbf{a}$ from a uniform distribution, the resulting distribution of control targets can differ between a delta action space and the base action space it approximates (e.g., ΔJP and JV). The clipping can also lead to more probability mass on the borders of the $\mathbf{v}_d$ space.

6.2 TASK-ACTION-SPACE SUITABILITY

The aim of this section is to illustrate that the choice of action space matters for robotic manipulation. We will illustrate that, by showing that some action spaces are not suitable for all tasks. Hence, we consider a simple 1D pushing example with one prismatic joint, as shown in Figure 6.2. The goal of this task is to push the blue cube into a target position. For the sake of the example, we assume a (non-delta) joint position action space as defined in the previous section. We further assume that moving the cube requires applying a force with a magnitude higher than 40N. Without loss of generality, we assume that $D = 0$, and that the joint is physically capable of applying forces higher than 40N.

Following equation (6.2), we can show that these task requirements impose a constraint of the form

$$K >= \frac{40}{q_{max} - q_{min}}, \tag{6.4}$$

where $q_{min} = -1$ and $q_{max} = 1$ as shown in Figure 6.2. Hence, the task imposes a constraint $K >= 20$ on the action space. While this constraint is not very hard to fulfill for the current task, it becomes problematic when the task requires the robot to be compliant with its environment or when a human is in the loop. Hence, we notice that if a task by definition requires different stiffness values (at least for some time), this requirement would be in conflict with the previous

constraint, making the task non-solvable with this choice of control space.

The main reason behind the appearance of this constraint is that the action space is not explicitly built for force control. Instead, the policy influences applied forces by outputting motion commands. This illustrates how the choice of action space can easily hinder task completion. While this example is intentionally simplified, similar results can be shown for more general settings with more DoF. The reason why such action spaces remain popular in the literature lies in the fact that most current robot learning tasks involve very light objects, smooth surfaces (low friction), and no human interaction. Such tasks can get away with almost any choice of action space. However, scaling robot learning to more dynamic tasks would require more careful considerations.

Furthermore, we notice that choosing an action space that controls a higher-order derivative might relax the constraint in equation (6.4). This is true for robots where the joint velocity limits are larger than the joint position limits, which would help increase the denominator of a constraint similar to the one in equation (6.4). As a result, such an action space might in many more cases be sufficient, despite it not explicitly allowing for force application as required by the task. This is another example of how changing the action space can influence the policy's capability of performing the given task.

6.3 EVALUATION METRICS‡

For each action space, we aim to assess its training performance, resulting sample efficiency, emerging properties such as usability in the real-world environment, and the sim-to-real gap it creates. Therefore, we propose multiple metrics to quantify these different aspects. For assessing training performance, we look at the **episodic rewards (ER)** in simulation. This metric can also show us the sample efficiency of the different action spaces.

To assess the emerging properties of each policy, we look at the number of times it violates robot constraints, such as the joint acceleration and jerk constraints. This is especially important since simulated environments rarely have any mechanisms for enforcing these constraints. In contrast, real-world robot control implementations can include rate limiters, which ensure that the control targets would not result in violations of these constraints. Hence, a policy trained in simulation without such mechanisms could learn to violate them for the sake of exploration. Implementing these mechanisms in simulation, on the other hand, can also hinder the policy training since these mechanisms typically break the Markov assumption. To quantify this

property, we use the **expected constraints violations (ECV)** defined as

$$\mathrm{ECV}(\pi) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\mathbb{1}\left(\sum_{c\in\mathcal{C}} c(\mathbf{s},\mathbf{a}) > 0\right)\right], \tag{6.5}$$

where $\mathbb{1}$ is an indicator function, $\mathcal{C}$ is the set of all constraints, and each constraint function $c(\mathbf{s},\mathbf{a})$ returns 1 for violated constrained and 0 otherwise. Furthermore, to measure the feasibility of the policy's actions in the environment, we report the **normalized tracking error (NTE)**,

$$\mathrm{NTE}(\pi) = \mathbb{E}_{\mathbf{s},\mathbf{a}\sim\pi}\left[\frac{|\mathbf{v}_{d,t} - \mathbf{v}_{t+1}|}{\mathbf{v}_{max} - \mathbf{v}_{min}}\right], \tag{6.6}$$

where $\mathbf{v}_{min}$ and $\mathbf{v}_{max}$ are respectively the lower and upper bounds of the control variable $\mathbf{v}$. NTE is useful for analysing why certain action spaces result in better transfer. A high value means that the policy outputs actions that are hard to achieve in one control step. This could create an additional sim-to-real gap since control targets that are not fulfilled in one step can be tracked differently in simulated and real environments due to the gap in dynamics. We also report the **task accuracy (ACC)** measured by the Euclidean distance to the goal. This metric gives a more detailed view of the performance of a policy than just the success rate.

To assess the sim-to-real gap of each action space we report the **offline trajectory error (OTE)** in configuration and task spaces. This metric measures the joint trajectory error when replaying, in simulation, the actions produced by the policy when queried in the real world,

$$\mathrm{OTE}(\pi) = \mathbb{E}_{\mathbf{a},\mathbf{q}_{real}\sim\mathcal{D}_{real}}|\mathbf{q}_{sim} - \mathbf{q}_{real}|, \tag{6.7}$$

where $\mathbf{a}$ and $\mathbf{q}_{real}$ are actions and joint configurations that are sampled from the dataset $\mathcal{D}_{real}$. The latter is collected by playing a policy π in the real world and $\mathbf{q}_{sim}$ is the joint configuration obtained when executing $\mathbf{a}$ in simulation in an open-loop fashion, i.e., without a policy.

6.4 EXPERIMENTS‡

We design experiments to understand the role of different characteristics of the action space on learning efficiency, sim-to-real gap, emerging properties, and sim-to-real transfer. Namely, we design experiments to answer the following questions:

(Q1) Does the choice of action space affect the exploration behavior during training in simulation?

(Q2) What properties naturally emerge due to the choice of action space?

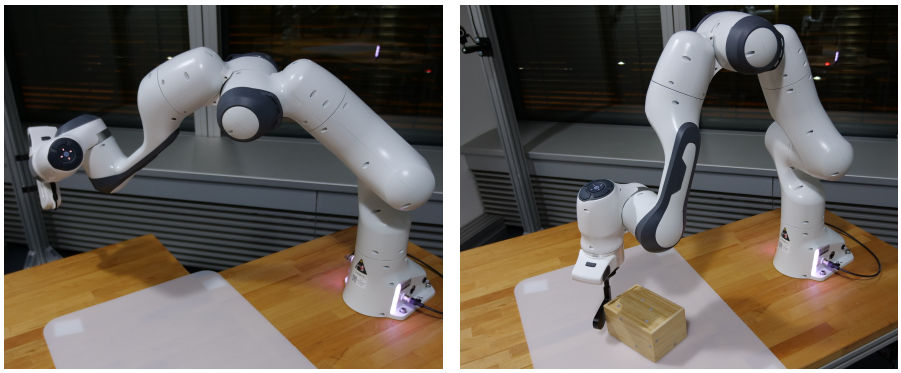


Figure 6.3: We show our real-world robot setup for the reaching (left) and the pushing (right) tasks.
This figure has previously been introduced in one of the author's previous publications [5].

(Q3) Does the action space affect the sim-to-real gap?

(Q4) Which action space characteristics are good for sim-to-real transfer?

(Q5) Is there a consistently best-performing action space?

To answer these questions we evaluate all action spaces on two arm manipulation tasks, using the 7-DoF Franka Emika Panda robot. The first task is goal reaching. At the beginning of each episode, a Cartesian goal is sampled in the workspace of the robot, and the policy needs to move the end-effector towards that goal. This task is ideal for studying the behavior of policies from all action spaces in the absence of force interactions with the robot's environment. The second task is object pushing. At the beginning of each episode, a goal position is sampled in a predefined area on the table. The policy needs to push a wooden box towards that goal. This task involves moving an external object. Unlike the reaching task, pushing requires physical interaction with the environment. It allows us to understand the reactive capabilities of each action space and whether it creates any sim-to-real gap that hinders the transfer of the interaction behavior. During policy training for pushing in simulation, we perform domain randomization on the box's friction and mass parameters. The real-world setup for both tasks is shown in Figure 6.3. In both tasks, the observation space of the policy consists of joint positions, joint velocities, end-effector Cartesian position, and the goal position. Additionally, in the pushing task the policy has access to the object's position and orientation. We train PPO policies in a simulated environment, using NVIDIA's Isaac Sim simulator [124].

The results discussed in this paper are based on 250 agents with different action spaces and different tasks. However, prior to the final training runs (of the 250 agents) we spent considerable time optimizing other hyperparameters, such as learning rates or different

Table 6.1: We compare the episodic reward (ER) obtained during training in simulation for all the studied action spaces. These include joint (J) and Cartesian (C) action spaces for position (P), velocity (V) and torque (T) control. They also include delta (Δ) spaces, with one-step (OI) and multi-step (MI) target integration schemes. The rewards are shown for reaching and pushing at multiple stages of the training process.
This table has previously been introduced in one of the author's previous publications [5].

Action	**Reaching (Epochs)**			**Pushing (Epochs)**		
space	**25**	**50**	**75**	**25**	**50**	**75**
JP	470 ± 7	478 ± 3	481 ± 3	15 ± 5	122 ± 82	208 ± 116
OIΔJP	474 ± 1	472 ± 1	464 ± 3	338 ± 35	376 ± 17	388 ± 21
MIΔJP	387 ± 14	425 ± 11	436 ± 6	155 ± 114	175 ± 150	114 ± 133
JV	438 ± 13	459 ± 4	455 ± 7	381 ± 17	400 ± 9	379 ± 39
OIΔJV	453 ± 3	457 ± 2	461 ± 3	332 ± 74	385 ± 27	359 ± 42
MIΔJV	455 ± 4	465 ± 3	465 ± 1	243 ± 157	383 ± 43	381 ± 16
JT	479 ± 2	476 ± 3	474 ± 3	71 ± 158	116 ± 182	187 ± 257
CP	456 ± 7	467 ± 2	465 ± 4	300 ± 157	322 ± 167	379 ± 34
OIΔCP	470 ± 2	467 ± 3	466 ± 3	365 ± 26	387 ± 13	395 ± 5
MIΔCP	409 ± 9	441 ± 2	442 ± 3	283 ± 29	311 ± 7	307 ± 21
CV	478 ± 2	476 ± 2	470 ± 2	410 ± 8	416 ± 8	423 ± 5
OIΔCV	471 ± 3	468 ± 4	464 ± 5	396 ± 15	422 ± 8	415 ± 13
MIΔCV	467 ± 2	467 ± 2	462 ± 3	393 ± 29	417 ± 12	414 ± 7

reward scales and terms, to guarantee consistency and a fair comparison over all action spaces. We matched the simulation to the real environment to the best of our knowledge, eliminating sim-to-real gaps where possible. This includes, for example, matching the stiffness of the impedance controllers, sharing implementations for each action space across both environments, and finding good velocity limits that allow for safe transfer of policies. This tuning process required training more than 20,000 agents in simulation and testing more than a 1500 agents in the real world.

After training, we evaluate the learned policies in both tasks, in simulation and in the real world. For reaching, we use a fixed grid of target goals that span the feasible workspace of the robot. For pushing, we randomly sample goal positions and use the object position from the previous episode as the initial one from the new run. We only manually reset the object's position whenever it gets outside of the predefined area it was trained to operate in or when the policy fails to push the object away from its starting location once. We introduced the latter intervention to avoid heavily skewing the data by randomly sampling a difficult-to-handle initial object position. In both training and evaluation, we run the simulation at 120 Hz and we use action repeat to work with the policy output at 60 Hz. On the real robot, we need an additional low-level safety controller running at 1 kHz, which also repeats the actions from the policy. In all action spaces except torque we additionally use a 5 Hz low-pass filter and a rate limiter to conform to motion limits given by the robot. We tried implementing the rate limiter in simulation as well, however, the policy learning deteriorated massively, which we attribute to the rate limiter breaking the Markov property. For the policy network, we use a 4-layer feed-forward neural network for all action spaces.

Does the choice of action space affect the exploration behavior during training in simulation?

We first examine the training performance in simulation. The results can be seen in the Table 6.1. The episodic rewards are not directly representative of the success rate of the policies. This can be seen when comparing the rewards from Table 6.1 to the success rates shown in Table 6.2 and Table 6.3. Therefore, we mostly focus on sample efficiency in the current analysis. Since different action spaces have different characteristics, we aim to understand the global effect of these characteristics on the sample efficiency during training. First, we compare Cartesian and joint action spaces. In the reaching task, both action space groups behave almost identically. However, in the pushing task, Cartesian action spaces seem to have an advantage in terms of sample efficiency and maximum reached reward, as can be seen in the top plot in Figure 6.4. This result is most likely due to the spatial nature of the pushing task, which gives Cartesian action spaces a natural advantage in exploration.

We next compare the order of derivative of the action space, i.e., positions vs. velocities vs. torques. Among joint action spaces, joint velocity and its derivatives have the overall best performance in both tasks. These action spaces show better sample efficiency and converge to higher reward regions than their counterparts in the joint action space group as can be seen in the bottom plot in Figure 6.4. JP reaches the highest reward in reaching but struggles massively in pushing. The same tendency can be observed for Cartesian action spaces, i.e., velocity action spaces perform better in terms of sample efficiency and final rewards. The joint torque action space is the fastest one to converge in the reaching task. However, it fails to solve the pushing task reliably within the same data budget.

We also compare base action spaces with the two different kinds of delta action spaces. We observe that multi-step integration delta action spaces consistently perform the worst, while one-step methods seem to have a slight advantage. This tendency is less pronounced in the velocity action spaces, with MIΔJV being consistently one of the best-performing joint action spaces in both tasks in simulation. Hence, the current simulation data does not conclusively favor any of these characteristics (non-delta, OI and MI).

Overall, we note that Cartesian velocity (CV) is the best-performing action space in simulation. The worst one is multi-step-integration joint position action space.

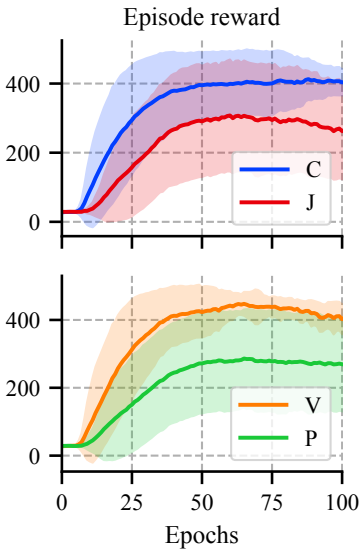


Figure 6.4: We show learning curves for the pushing task and aggregate action spaces according to some of the characteristics named above. *This figure has previously been introduced in one of the author's previous publications [5].*

What properties naturally emerge due to the choice of action space?

After training the policies in simulation we evaluate them on a real robotic setup. For the pushing task, we were not able to obtain joint torque policies that are safe to run in the real world without damaging the robot. We made multiple attempts to produce safely deployable joint torque policies, for instance, by introducing different penalties or increasing the policy's control frequency. Despite all efforts, all joint torque policies were very jerky or aggressive when deployed on the real robot. The main difficulty was to obtain policies that are safe to deploy in a task where the end-effector needs to remain in close proximity to a rigid surface, e.g., the table. Therefore,

Table 6.2: Sim-to-real transfer evaluation for the reaching task. We evaluate the success rate (SR) in simulation and the real-world environment, the task accuracy (ACC), the expected constraints violations (ECV), and the offline trajectory error (OTE).
This table has previously been introduced in one of the author's previous publications [5].

	SR (sim) ↑	**SR** (real) ↑	**ACC** [cm] ↓	**ECV** ↓	**OTE** [rad] ↓
JP	**100 ± 0**	6 ± 6	1.76 ± 0.14	100 ± 0	0.22 ± 0.01
OIΔJP	**100 ± 0**	86 ± 9	1.59 ± 0.16	14 ± 1	0.45 ± 0.01
MIΔJP	**100 ± 0**	69 ± 2	2.14 ± 0.13	**0 ± 0**	**0.03 ± 0.00**
JV	**100 ± 0**	**100 ± 0**	**1.17 ± 0.11**	0 ± 0	**0.03 ± 0.00**
OIΔJV	**100 ± 0**	97 ± 5	1.35 ± 0.13	13 ± 11	0.18 ± 0.01
MIΔJV	**100 ± 0**	93 ± 9	1.98 ± 0.08	37 ± 37	**0.03 ± 0.00**
JT	**100 ± 0**	64 ± 27	1.97 ± 0.25	22 ± 6	0.52 ± 0.03
CP	99 ± 2	25 ± 4	1.71 ± 0.29	26 ± 9	0.16 ± 0.01
OIΔCP	89 ± 2	44 ± 17	2.02 ± 0.32	21 ± 1	0.27 ± 0.01
MIΔCP	**100 ± 0**	68 ± 2	1.85 ± 0.08	13 ± 4	0.28 ± 0.01
CV	**100 ± 0**	43 ± 13	1.76 ± 0.41	12 ± 2	0.35 ± 0.02
OIΔCV	97 ± 2	24 ± 2	2.13 ± 0.13	17 ± 3	0.40 ± 0.01
MIΔCV	99 ± 2	58 ± 8	1.82 ± 0.08	4 ± 0	0.13 ± 0.01

Table 6.3: Sim-to-real transfer evaluation for the pushing task. We evaluate the success rate (SR) in simulation and the real-world environment, the task accuracy (ACC), the expected constraints violations (ECV), and the offline trajectory error (OTE).
This table has previously been introduced in one of the author's previous publications [5].

Action space	**SR** (sim) ↑	**SR** (real) ↑	**ACC** [cm] ↓	**ECV** ↓
JP	87 ± 12	4 ± 0	4.23 ± 0.66	100 ± 0
OIΔJP	99 ± 3	48 ± 12	3.16 ± 0.46	98 ± 2
MIΔJP	90 ± 7	36 ± 3	3.52 ± 0.36	**3** ± **1**
JV	97 ± 5	90 ± 5	2.04 ± 0.32	66 ± 9
OIΔJV	99 ± 3	88 ± 13	**1.56** ± **0.13**	75 ± 4
MIΔJV	**100** ± **0**	**96** ± **4**	1.66 ± 0.11	55 ± 25
JT	40 ± 49	- ± -	- ± -	89 ± 21
CP	**100** ± **0**	57 ± 9	2.34 ± 0.39	92 ± 3
OIΔCP	99 ± 3	55 ± 17	2.86 ± 0.49	35 ± 16
MIΔCP	**100** ± **0**	73 ± 7	2.36 ± 0.14	99 ± 0
CV	99 ± 3	73 ± 9	2.69 ± 0.58	93 ± 10
OIΔCV	99 ± 3	61 ± 19	2.67 ± 0.30	99 ± 0
MIΔCV	86 ± 5	74 ± 4	2.84 ± 0.45	6 ± 2

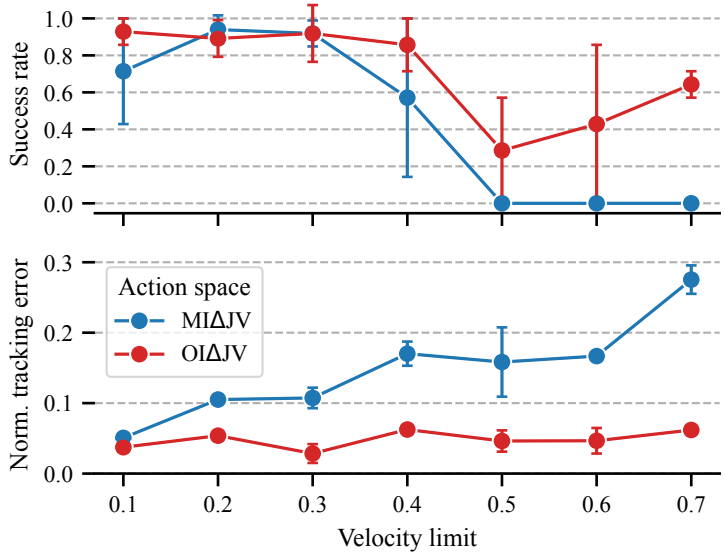


Figure 6.5: We show the the robustness of delta action spaces to the velocity limit hyperparameter in the pushing task on our real setup. We compare one-step and multi-step integration. The errorbars indicate one standard deviation of the respective variables.
This figure has previously been introduced in one of the author's previous publications [5].

we exclude the joint toque action space from our real-world pushing experiments. The reaching task was less safety-critical. Hence, we managed to evaluate torque policies on real-world reaching. For all other action spaces, we look at their sim-to-real transfer capabilities. In Table 6.2 and Table 6.3, we report the metrics introduced in Section 6.3 to quantify the sim-to-real gap and the performance in the real world. One common challenge when learning manipulation skills is to obtain smooth policies that do not violate the velocity, acceleration, and jerk constraints of the robot. Based on the results in Table 6.2 and Table 6.3, we observe that different action spaces yield different ECV metrics. JV and its derivatives (OIΔJV and MIΔJV) have on average the lowest ECV score in both tasks. Vanilla JP results in the highest possible ECV. This means that deploying this action space in the real world would always result in some form of constraint violation. In the absence of safety mechanisms (such as rate limiters and low-pass filters) these violations can be very harmful. If such mechanisms are implemented, this behavior would increase the sim-to-real gap further because of how these violations trigger the safety mechanisms. In contrast, MIΔJP, seems to have the lowest ECV, which was also evident when evaluating this action space in the real world.

Does the action space affect the sim-to-real gap?
First, we look at the offline trajectory error in the reaching task, which gives us a proxy measure of the sim-to-real gap. We observe that the OTE varies tremendously from one action space to another, despite

the fact that the data used to compute this measure are based on the same starting and goal positions in all action spaces. This confirms that *the choice of action space does indeed contribute to the sim-to-real gap*. Looking closer at the results, we observe that JT exhibits the highest OTE. This is due to the behavior of this action space being dictated solely by the dynamics of the robot, which is different in simulation and in the real world. Unlike all the other action spaces, JT does not include any feedback loops outside of the policy. Based on this result, one would expect that more feedback loops should help reduce the sim-to-real gap. However, the opposite can be seen in the data. For instance, Cartesian action spaces have on average one additional feedback loop compared to the joint action spaces. Their OTE is, however, higher on average. This is due to the fact that feedback loops have different effects in simulation than in the real world. In turn, this means that a good, highly-reactive feedback loop is beneficial to overcome the dynamics gap, but adding more could potentially contribute further to the sim-to-real gap. Comparing OIΔ and MIΔ action spaces we notice that the latter consistently have a smaller OTE. Their OTE is even smaller than the corresponding base action spaces. This effect is potentially due to the integral term embedded in these action spaces. Executing the same actions results in the same final goal given to the lower-level feedback loops, which is unique compared to all other action spaces.

Which action space characteristics are good for sim-to-real transfer?

We compare the success rates in simulation and the real world. We observe that *the success rate in simulation is not directly reflected in the real world*. This is clear when comparing the ordering of success rates in both domains. Furthermore, we notice that certain action space characteristics are clearly advantageous for sim-to-real transfer. For instance, velocity-based action spaces tend to keep a high success rate when transferred to the real world. In contrast, position-based action spaces do not typically transfer well. This is especially the case for JP, which loses almost all of its performance when transferred. These results are consistent across both studied tasks. In addition, velocity-based action spaces are on average more accurate in fulfilling the task as can be seen when comparing the ACC score in Table 6.2 and Table 6.3.

Additionally, we can see that delta action spaces transfer better than their corresponding base spaces. The difference between OI and MI delta action spaces depends on the task and the exact variants of the action space. However, OIΔ action spaces required less tuning than MIΔ spaces and have shown to be more robust to the choice of hyperparameters. This can be seen in Figure 6.5. When varying the velocity limits for ΔJV action spaces, OIΔJV showed to be less sensitive to the chosen value. We attribute this to the lower tracking error

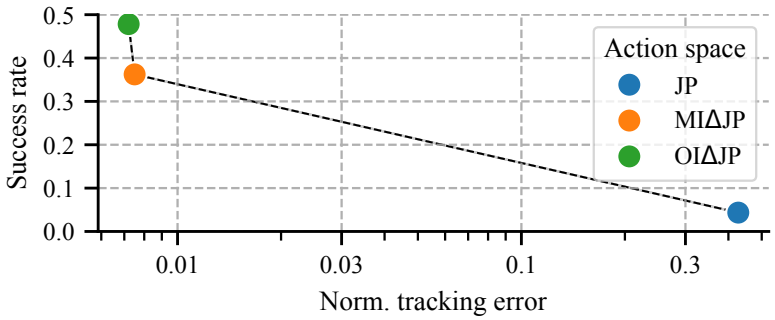


Figure 6.6: We compare joint position-based action spaces in the pushing task and show their influence on the normalized tracking error and the resulting effect on success rate in the real robot setup. *This figure has previously been introduced in one of the author's previous publications [5].*

of this action space as shown in the bottom plot in Figure 6.5. In general, OIΔ action spaces naturally lead to a lower tracking error since they integrate the policy actions into control targets based on the current feedback of the system. In contrast, MIΔ action spaces integrate the policy actions into control targets based on the previous control target. This in turn leads to a bigger gap between the control target and the state of the system, and hence a larger tracking error. While a lower NTE helps make JV action spaces more robust to hyperparameters, it has an even larger effect on the best possible transfer performance in JP action spaces, as shown in Figure 6.6. This tendency is also evident when comparing the base spaces, as shown in Figure 6.7. Finally, joint velocity spaces show better transfer than Cartesian velocity action spaces, while the opposite is true for position-based action spaces.

In summary, our data shows that two characteristics mostly influence the transfer capability of an action space. The first one is the order of the derivative of the control variables. With the exception of joint torque control, which is problematic for other reasons, *an action space that controls a higher order derivative transfers better.* The second characteristic is the emerging tracking error of the action space's control variable(s). Our data strongly indicates that *an action space which yields or naturally limits the tracking error transfers better.* This last property can be enforced by the action space design, for instance, by reducing the magnitude of jumps of the corresponding control targets. The latter can be controlled by the scaling of the actions in delta action space, or by increasing the stiffness if the task allows. For action spaces that do not naturally allow for controlling this property, one could attempt to enforce a smaller NTE by means of rewards/penalties on the actions' magnitude and smoothness. However, based on our experience, the tuning process for the resulting additional hyperparameters (of such reward terms) can be very difficult.

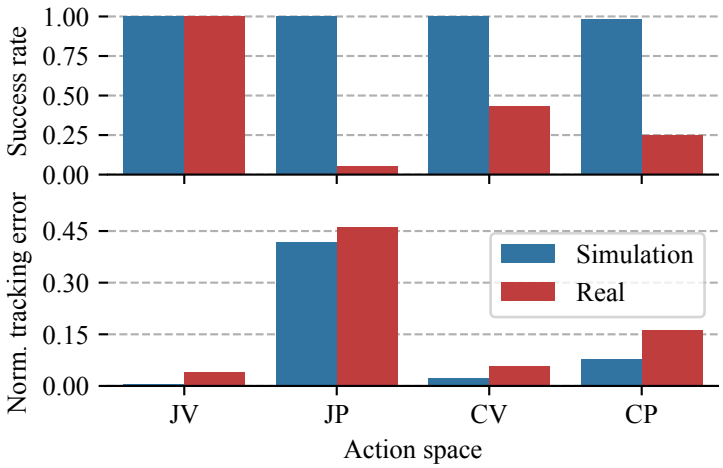


Figure 6.7: We show the influence of the normalized tracking error on sim-to-real transfer in terms of success rate for non-delta action spaces in the reaching task.
This figure has previously been introduced in one of the author's previous publications [5].

Is there a consistently best-performing action space?
Overall, we notice that *joint velocity action spaces seem to have the best performance in sim-to-real transfer*. These action spaces have on average the lowest OTE, ACC and ECV and transfer the best to the real world. They also required the least tuning to work, making them the most suited action spaces for manipulation learning (among the studied options). This result is consistent with our previous findings concerning favorable characteristics, i.e., JV-based action space benefits from a higher-quality feedback loop due to their configuration-space control, and can more easily generate high forces for interaction than position-based action space due to controlling a higher-order derivative.

6.5 DISCUSSIONS

Learning manipulation skills can be a very challenging process. A manipulation system needs to perceive its environment and act in it. In learning-based manipulation systems, the action interface is defined by the action space of the policy. By embedding well-established control principles from the robotics literature, we could construct a large variety of action spaces for this domain. However, the choice of action space fundamentally changes the learning problem and can hence make or break the manipulation system. Besides the effect this choice has on learning performance, it strongly influences the capabilities of the learned policies. In addition, this choice can completely change the course of sim-to-real transfer.

This chapter discussed the most popular choices in the literature and studied their performance empirically. At its center is an empirical analysis of manipulation action spaces, their characteristics, and emerging properties. Additionally, our analysis attempts to build a deeper understanding of the relationships between characteristics and emerging properties, with the aim to provide clear recommendations for future methods. This analysis is based on a carefully designed study with 13 different action spaces and 2 manipulation tasks. The data strongly indicates that certain action space characteristics are very favorable for the overall behavior and performance of manipulation policies. Namely, we show that velocity-based action spaces are substantially better for learning and sim-to-real transfer in comparison to torque or position-based methods. Furthermore, we show that keeping a low tracking error can improve the transfer of policies from simulation to the real world. More generally, our results indicate that implementation details and tuning of controllers also play a big role in learning and transferring robotic policies.

6.5.1 *Limitations*

This study and the resulting analysis are limited to the considered action spaces and tasks. We chose these two in a way that would represent a large variety of options, but in no way do we claim to have answers that are general enough for robotics or even manipulation. In addition, the study only considered one robotic manipulator. While this choice should not be very limiting to the general truth of our findings, it might still slightly bias the analysis. Besides the hardware difference, different robots have different software stacks, communication protocols, and safety mechanisms. All of these aspects could affect the analysis. However, the current study setup was already quite expensive in terms of training costs, robot damage, human effort, and time. A larger study with more robots, more tasks, and different control implementations is only feasible in the context of a larger collaboration.

6.5.2 *Outlook*

The aim of this study was to better understand different aspects of action space design and how they influence manipulation learning. However, besides scientific curiosity, this gained knowledge can be useful for future applications and future research efforts in robot learning. Another important goal of this work was to shed more light on the often ignored action aspect of robot learning and its important role. Future methods can build on our findings to build novel action spaces, perhaps ones that are more suitable for manipulation or maybe more generally applicable. We focused on the sim-to-real

transferability of policies not only due to the emerging popularity of this paradigm, but also because it presents a wider framework to understand different aspects of the problem. With simulations becoming more powerful and more affordable, the role of this paradigm could become more important in the coming years. This is especially true with the emergence of foundational robotics models. The latter typically rely on large datasets for training or assume access to robotics skills. Sim-to-real transfer could cater, to some (limited) extent, to both the data and skills requirements. Future work on sim-to-real transfer can readily apply our findings or build on top of it.

7

INDUCTIVE BIASES IN THE ACTIONS SPACE

The previous chapter introduced action spaces that use task and configuration-space controllers to translate policy actions into robot joint torques. While such action spaces can be very effective for learning manipulation skills, it is still possible to abstract more properties of a manipulation skill. When building such abstractions, we should aim at offloading complexity from the policy to the action space. However, to avoid over-specifying an action space (i.e., making it specific for some tasks), such abstractions should handle variables that are common to all manipulation tasks.

For example, a manipulation policy typically handles the motion of the robot and its interaction with the environment. The robot's motion can be defined in either a task or configuration space. It typically also adheres to constraints in either space. Therefore, a good manipulation policy should be able to fully control these variables to enable the robot to perform any given task. The action spaces presented in the previous chapter fulfill these requirements. However, they still require the policy to control some unnecessary variables. For example, if the robot is supposed to reach some object before manipulating it, the policy does not necessarily need to control each robot configuration or end-effector position along the way of reaching the object. This added complexity makes the training of such a policy sample inefficient. Instead, the policy could only define a single target position and some parameters that define the trajectory to reach this position in a way that respects the task constraints.

However, complexity can also be added by means of under-specification of the action space, not just its over-specifications. For example, imagine a task that involves the robot interacting with its environment or any kind of external force acting on its links. For the sake of argument, we can explicitly consider the task of lifting a heavy object. A policy trained to only output motion targets (e.g., joint position targets) has no direct means of reacting to the resulting gravity forces acting on the object. Such a policy would then need to output motion targets that would lead the underlying controllers to compensate for these external forces. Learning such a behavior is quite complex. It would require the policy not only to understand the interaction but also to find a way to leverage the low-level controllers to interact with its environment. In this case, extending the action space to include a desired output force or stiffness can reduce the complexity of the policy despite the fact that it increases the action's dimensionality.

The choice of action space is important not only for reducing the policy complexity but also for ensuring its feasibility. If we once again

consider the previous example, one could argue that certain reaction forces needed to achieve certain tasks cannot be achieved simply by specifying distant motion targets. In the lifting example, the object could be too heavy and could require a large force at the hands. A policy acting only in motion space would need to produce very far-away motion targets to force the robot to apply such forces. In most cases, the motion targets are bounded on the basis of the robots' kinematics and its workspace reach. The only way to allow such a policy to perform the task would be to allow it to output motion targets beyond its actual capabilities. Hence, the only way to make use of such an action space in this case would require relaxing the motion constraints beyond the robot's actual limits. Such a solution would allow the policy to successfully complete the task. However, it poses safety concerns. The ideal action space for robot manipulation is neither under nor over-specified. It should allow the policy to control just the right variables. This problem is very challenging, especially when attempting to design an action space that is suitable for all or at least a wide range of manipulation tasks. Using the right inductive biases, we can build such action spaces and simplify policy learning.

7.1 OBSTACLE-FREE MOTION GENERATION

In this chapter, the focus is on the motion aspect of an action space. While interacting with its environment, a robot should be able to move its body in a way that enables the execution of the task, respects the task constraint, and does not harm the robot or its environment. In practice, the task makes certain states more favorable for success and imposes constraints on the robot's motion. One of the biggest challenges in robot motion generation is to avoid collisions. In a well-structured environment, collision avoidance can be ensured by carefully designing collision-free trajectories, or simplifying the motion generation process by leveraging knowledge about the obstacles' shapes and positions. Obstacle avoidance is a fundamental challenge in robotic manipulation for which many solutions have been proposed over the years. These can be categorized into local and global methods [93]. Local algorithms only adapt the manipulator's behavior in the presence of obstacles. Potential fields [94] are a famous example of such methods. Local solutions typically employ reactive closed-loop control to generate local motion. These methods usually have a low complexity and can hence be run in the inner low-level control loop of the manipulator [65]. This ensures a level of responsiveness that is needed for critical scenarios, such as safe human-robot interaction. However, these approaches assume Euclidean geometry in the task space of the manipulator [95] and only use internal geometry of the kinematic chain [41]. Despite their simplicity and versatility, purely reactive methods often culminate in undesirable behaviors such as instability or oscillation [157]. On the other

hand, global planning-based, differential geometric approaches aim to model the intrinsically non-euclidean task space of the robot [158]. This thread of research has seen significant progress recently and primarily uses optimization to generate nonlinear trajectories [200]. Nonetheless, planning-based methods are notoriously computationally expensive and lack the reactiveness of local methods. To avoid obstacles, these methods require knowledge about obstacles (e.g., keypoints) which is crucial for autonomous applications in unstructured, dynamic environments. This need for semantic understanding of the environment has sprouted a variety of research on vision-based obstacle avoidance, which is especially prominent in mobile robotics [63].

On the other hand, motion planning or generation in unstructured environments can be very challenging. In such environments, the robot should be capable of reacting to unknown and previously unseen obstacles in real-time. An important aspect when approaching such systems is to have a robust perception pipeline. This means that the action space should be designed in a way that ensures that the actions can be inferred given the perception system's output. In addition, the action space can handle trivial aspects of motion generation such as reaching a target in the absence of obstacles. When the policy encounters a novel obstacle for the first time during training, it needs to react directly and alter the robot's motion to avoid a collision. Obstacles can be numerous and potentially dynamic. Therefore, a manipulation action space should be designed in a way that allows the policy to handle such occurrences.

7.2 ABSTRACTING MOTION

The most salient attributes of robotic manipulators are best expressed in terms of differential geometry [96]. More specifically, motion generation and control can be seen as the problem of transforming desired behavior from one or multiple smooth manifolds — representing the task space $\mathcal{T}$ — to another smooth manifold $\mathcal{C}$, the configuration space. These manifolds are related by a differential map $\phi : \mathcal{C} \to \mathcal{T}$, called the task map. By equipping these manifolds with a Riemannian metric $\mathbf{M}$, we can elegantly relate notions like angles and distances to design a curve $q(t) \in \mathcal{C}$ which implements the desired behavior of $\mathcal{T}$. The metric tensor captures the notion of distances and angles between points in the configuration space. Its representation can encode constraints in the environment. The choice of this metric can significantly impact how the robot navigates through the $\mathcal{C}$-space. For instance, one could modify the Riemannian metric in such a way that it reflects the influence of obstacles. This often involves shaping the metric in a way that increases distances in regions close to obstacles, effectively making those regions more costly to traverse. This means that the metric tensor is adjusted to reflect the local curvature and distances based on the presence of obstacles. When planning the robot's motion, the

modified Riemannian metric is used to guide the planner. As the metric encodes the obstacle information, the planner will naturally try to find paths that avoid the high-cost regions associated with obstacles. This approach is particularly useful when dealing with complex and dynamic environments. A motion-focused action space could leverage this metric to encapsulate motion-information in a single entity (i.e., the Riemannian metric). The explicit way to achieve this goal is to define the action space to be the parameters ζ of a parametrized Riemannian metric $\mathbf{M}_\zeta(\mathbf{x}, \dot{\mathbf{x}})$. In that case, the policy would need to learn to map its observations into these parameters. Given these parameters, we can plan the robot's motion based on the resulting metric. Depending on the frequency of the policy, we can control the reactivity of the robot. The proposed action space seems great in theory. However, in practice, due to the large number of parameters in ζ, training such as mapping based on RL would require a lot of samples, especially when using image observations. We can reduce the number of parameters needed to present the metric by using a Cholesky decomposition parameterization. However, the number of parameters to properly represent motion in dynamic environments remains high and hence the action space would have a problematically high action dimensionality. Additionally, such an action space can make the policy search very inefficient for static environments. This is because static environments would require one single metric for each obstacle configuration. Hence, in practice, the policy would need to output a single action for the whole episode. This would exacerbate the sample complexity of the task despite being simpler than dynamic obstacle avoidance.

Instead, we could build an action space that implicitly leads to similar behavior. For that, in this work, we propose a framework to learn reactive motion policies with an action space constructed using Riemannian motion policies (RMP) [157] and RMPflow [41]. An RMP is a framework for motion control, grounded in the geometry of a Riemannian manifold. The generation of these policies is an intricate process that hinges not only on the potential field within the space but also on the underlying curvature of the space itself [157]. Consider an arbitrary m-dimensional manifold $\mathcal{M}$ with generalized coordinates $\mathbf{x} \in \mathbb{R}^m$. In its canonical form, an RMP is defined as $(\mathbf{a}, \mathbf{M})_\mathcal{M}$, where $\mathbf{a} : \mathbf{x}, \dot{\mathbf{x}} \mapsto \ddot{\mathbf{x}}^d \in \mathbb{R}^m$ represents a second-order dynamical system mapping $\mathbf{x}$ and $\dot{\mathbf{x}}$ to desired accelerations $\ddot{\mathbf{x}}^d$, and $\mathbf{M} = \mathbf{M}(\mathbf{x}, \dot{\mathbf{x}}) \in \mathbb{R}^{m \times m}$ is a Riemannian metric which varies smoothly with the state $(\mathbf{x}, \dot{\mathbf{x}})$. We can interpret $\mathbf{M}$ as an inertial matrix which also defines the weight of the RMP when combined with others. In its natural form, an RMP is defined as $(\mathbf{f}, \mathbf{M})_\mathcal{M}$, where $\mathbf{f}$ indicates the map from position and velocity to the desired force. The force map $\mathbf{f}$ in the natural form and the acceleration map in the canonical form has the relation $\mathbf{f} = \mathbf{Ma}$. The natural form of an RMP is commonly used when space transfor-

mations are to be applied, due to the lower computational complexity of such operations using this form.

The RMP framework also provides `push`, `pull`, and `add` operators. `push` and `pull` can transform an RMP defined in a certain task space into another, based on the task map ϕ and its Jacobian $\mathbf{J}$. For example, when transforming the robot's joint positions $\mathbf{q}$ and velocities $\dot{\mathbf{q}}$ into the end-effector poses $\mathbf{p}$ and velocities $\dot{\mathbf{p}}$ respectively, ϕ would then correspond to the forward kinematics and $\mathbf{J}$ to the Jacobian of the manipulator. As for `add`, it is used to compose RMPs defined in the same task space into one policy.

RMPflow is a policy synthesis framework [41]. In this method, RMP-tree is introduced as a tree structured computational graph. Each node in such a tree represents a Riemannian manifold and is equipped with an RMP. The root node of the tree describes the configuration space $\mathcal{C}$ of the robot where the global joint space policy is defined. The leaf nodes represent task spaces, where the designed or learned local policies are defined.

The RMPs in this framework are characterized by a set of geometric dynamical systems (GDSs). Consider a manifold $\mathcal{M}$ with generalized coordinate $\mathbf{x} \in \mathbb{R}^m$, a GDS defined on such manifold can be expressed as

$$(\mathbf{G}(\mathbf{x}, \dot{\mathbf{x}}) + \Xi_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}))\ddot{\mathbf{x}} + \xi_{\mathbf{G}}(\mathbf{x}, \dot{\mathbf{x}}) = -\nabla\Phi(\mathbf{x}) - \mathbf{B}(\mathbf{x}, \dot{\mathbf{x}})\dot{\mathbf{x}}, \tag{7.1}$$

where $\mathbf{G} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ is referred to as the metric matrix, $\mathbf{B} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}_+^{m\times m}$ as the damping matrix and $\Phi : \mathbb{R}^m \to \mathbb{R}$ as the potential function. Additionally, $\Xi_{\mathbf{G}} := \frac{1}{2}\sum_{i=1}^{m} \dot{x}_i \partial_{\dot{\mathbf{x}}} \mathbf{g}_i$ and $\xi_{\mathbf{G}} := \overset{\mathbf{x}}{\mathbf{G}}\dot{\mathbf{x}} - \frac{1}{2}\nabla_{\mathbf{x}}(\dot{\mathbf{x}}^{\mathbf{G}}\dot{\mathbf{x}})$ are curvature terms induced by the metric $\mathbf{G}$, where $\mathbf{g}_i$ denotes the i-th column of $\mathbf{G}$ and $\overset{\mathbf{x}}{\mathbf{G}} := [\partial_{\mathbf{x}}\mathbf{g}_i\dot{\mathbf{x}}]_{i=1}^{m}$. Based on the GDS formulation, the RMP metric term $\mathbf{M}$ is defined as $\mathbf{M} := \mathbf{G} + \Xi_{\mathbf{G}}$. The forcing term of RMP can be calculated by moving the curvature term $\xi_{\mathbf{G}}$ to the right hand side of equation (7.1), we then obtain

$$\mathbf{f} = (\mathbf{M}\ddot{\mathbf{x}}) = -\xi_{\mathbf{G}} - \nabla\Phi - \mathbf{B}\dot{\mathbf{x}}. \tag{7.2}$$

RMP-algebra introduces the `pushforward`, `pullback` and `resolve` operators used to construct the policy generation process. The goal of this process is to generate the desired accelerations $\ddot{\mathbf{q}}^d$ in the root node of the tree (robot configuration space) based on $\mathbf{q}$ and $\dot{\mathbf{q}}$. The `pushforward` operator is applied to transform the current joint state $(\mathbf{q}_t, \dot{\mathbf{q}}_t)$ into states in all leaf nodes, namely $\{(\mathbf{x}_{it}, \dot{\mathbf{x}}_{it})\}_{i=1}^{N}$, where i is the number of the leaf nodes. After the state information is obtained at the leaf nodes, the value of the desired leaf nodes' forces and metrics $\{(\mathbf{f}_{it}^d, \mathbf{M}_{it}^d)\}_{i=1}^{N}$ can be calculated according to equation (7.1). The obtained values are then transformed into the desired force and metric $(\mathbf{f}_t^d, \mathbf{M}_t^d)$ in the root node using the `pullback` operator. Consequently, the `resolve` operator is used to convert the desired force $\mathbf{f}_t^d$

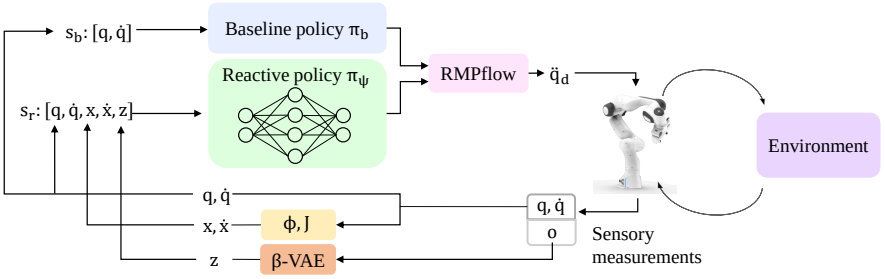


Figure 7.1: System Overview: At a certain time step, the system receives a visual input $\mathbf{o}$, joint angle measurements $\mathbf{q}$ and joint velocities $\dot{\mathbf{q}}$. The image $\mathbf{o}$ is then encoded using a β-VAE encoder to produce the visual latent $\mathbf{z}$. Using the robot kinematics ϕ and jacobian $\mathbf{J}$, we obtain the end-effector position $\mathbf{x}$ and velocity $\dot{\mathbf{x}}$ from $\mathbf{q}$ and $\dot{\mathbf{q}}$ respectively. Subsequently $\mathbf{q}$ and $\dot{\mathbf{q}}$ are fed into a baseline policy to produce a user-defined behavior. Simultaneously, we feed all available information $\mathbf{s}_r$ into a reactive policy π_δ. The latter produces a reactive behavior dependent on the objects in the environment. Both outputs are then composed together based on the RMPflow framework, to produce a desired joint acceleration $\ddot{\mathbf{q}}_d$. This acceleration is then fed into the robot controller.
This figure has previously been introduced in one of the author's previous publications [4].

into the desired joint acceleration $\ddot{\mathbf{q}}_t^d$. For further information about this framework, we refer the readers to the original paper [41].

Using these two frameworks, we propose an action space that simplifies learning motion policies for manipulation. The framework can be shown in Figure 7.1. Generally, the action space consists of two internal policies. The first is a hand-crafted baseline policy. The aim of this part is to guide the robot's motion generation towards its target. This policy can be as simple as a point attractor. The second is a reactive policy which can correct the baseline behavior depending on the perceived situation. This policy reacts to obstacles based on its knowledge of the end-effector position, current robot configuration, and visual input. These two policies are not learned but instead hand-designed and embedded in the action space. The learned RL policy outputs control variables that guide the behavior of those policies. The next two sections explain the details of the baseline and reactive policies, and what they expect from the learned RL policy.

7.2.1 *Baseline policy‡*

As previously mentioned, our primary task is goal reaching. For this purpose, we define a simple baseline RMP $\mathcal{R}_b = (\mathbf{f}_b, \mathbf{M}_b)_{\Omega_b}$ for goal reaching based on [41, 157], where Ω_b is a 3-dimensional manifold in which the baseline policy is defined. Namely, for $\mathbf{d} \in \Omega_b$, $\mathbf{d}$ measures the distance between the end-effector and the goal along the x, y and z axes. Specifically, we use a GDS to define the RMP in Ω_b. We first

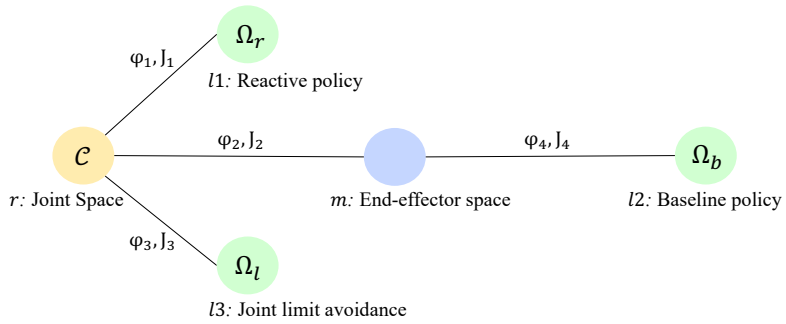


Figure 7.2: Structure of our RMP tree for combining multiple policies. *This figure has previously been introduced in one of the author's previous publications [4].*

define the metric $\mathbf{G}_b$ to be an identity matrix, which eliminates the curvature terms in the GDS to $\mathbf{M}_b := \mathbf{I}$ (based on the definitions in section 7.2), where $\mathbf{I}$ is the identity matrix. To ensure stability, we define the damping matrix as $\mathbf{B}_b := w\mathbf{I}$, where w is a vector that consists of positive constants close in value to zero. Moreover, we define the gradient of the potential field $\nabla\Phi(\mathbf{d}) := \mathbf{d}$. Substituting these values in equation (7.2), we obtain the GDS with the form

$$\mathbf{I}\ddot{\mathbf{d}} = -\mathbf{d} - w\mathbf{I}\dot{\mathbf{d}}. \tag{7.3}$$

This would result in the forcing term $\mathbf{f}_b = -\mathbf{d} - w\mathbf{I}\dot{\mathbf{d}}$ according to equation (7.2).

7.2.2 *Reactive policy*‡

To enable the obstacle avoidance behavior, we define a reactive RMP $\mathcal{R}_r = (\mathbf{f}_r, \mathbf{M}_r)_{\Omega_r}$ in a 7-dimensional manifold Ω_r. For $\mathbf{d}_q \in \mathbb{R}^7$ in Ω_r, $\mathbf{d}_q$ measures the distance between the current joint configuration $\mathbf{q}$ and the desired joint configuration $\mathbf{q}_g$. We set the metric, the damping matrix and the potential field of this RMP similar to the ones in the previous section. According to equation (7.2), the resulting forcing term is

$$\mathbf{f}_r = -\mathbf{d}_q - w\mathbf{I}\dot{\mathbf{d}}_q. \tag{7.4}$$

We are then concerned in learning a mapping π_δ from an input state $\mathbf{s}_r = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{z}]$ to an output action $\mathbf{a} = \mathbf{d}_q$, where $\mathbf{x}$ corresponds to the end-effector position, and is obtained from $\mathbf{q}$ using the kinematics ϕ_1 of the manipulator, and $\dot{\mathbf{x}}$ is obtained from $\dot{\mathbf{q}}$ via the Jacobian $\mathbf{J}_1$. As for $\mathbf{z}$, it is the latent representation of a given image $\mathbf{o}$ (as in section 7.2.3). We model π_δ as a multilayer perceptron (MLP) with two hidden layers. It is then trained via residual RL [85] with the baseline policy from section 7.2.1. For this purpose we use the Twin Delayed Deep Deterministic policy gradient algorithm [54]. Given the

end-effector position $\mathbf{x}$, the goal position $\mathbf{x}_g$, and the action $\mathbf{a}$ per step, the reward is defined as

$$r := r_{\text{collide}} + r_{\text{goal}} + r_{\text{dist}} + r_{\text{control}}, \tag{7.5}$$

where r_{collide} is a negative reward of -10, which punishes the robot when colliding with an obstacle, r_{goal} is a positive incentive of 5 given only when the robot's end-effector reaches the goal, $r_{\text{dist}} = -1.6\|\mathbf{x} - \mathbf{x}_g\| + 0.75$ is the distance reward which rises linearly when the distance between the end-effector and the goal decreases, and $r_{\text{control}} = -0.05\|\mathbf{a}\|$ which punishes large actions. This last term encourages the policy to diverge from the baseline policy only in cases of possible collision. The framework is not sensitive to changes in the values used in the reward function as long as the general relation between the terms is preserved. Namely, the distance reward is an important signal for training. When its magnitude is large, it leads to a local solution where the collision reward is neglected. Additionally, we aim to ensure that the episode rewards for goal-reaching and collision avoidance are not dominated by the control reward, hence the small weight 0.05. This combination of rewards encourages the resulting policy to move along the geodesic of the corresponding Riemannian Manifold. Furthermore, our MDP has a finite horizon with episodes ending at a specified maximum number of steps or whenever the robot collides with an obstacle. During early exploration stages, most episodes are unsuccessful and end up with collision. Consequently, only few successful trials are recorded. We use Prioritized Experience Replay [163] in order to use data from such trials more efficiently.

7.2.3 *Overall System*

In addition to the baseline and reactive policies defined in the previous sections, we use an additional RMP $\mathcal{R}_l = (\mathbf{f}_l, \mathbf{M}_l)_{\Omega_l}$ for joint limit avoidance similar to the one in [157]. More information concerning this policy can be found in Appendix B.2.1. All three policies are then combined together based on the tree in Figure 7.2. The node l_1 uses the RMP $\mathcal{R}_r$ from section 7.2.2, l_2 uses the $\mathcal{R}_b$ from section 7.2.1, and l_3 uses the joint limit avoidance RMP $\mathcal{R}_l$. Consequently, RMP-algebra is applied to compute the desired joint acceleration $\ddot{\mathbf{q}}_d$. We treat the motion trajectories given by the RMP as desired trajectories, and track them using an impedance control with suitable collision handling algorithms guaranteeing contact force and torque thresholds. At the policy level, when a collision takes place during task exploration and execution, it is detected safely, the episode terminates, and the environment returns a penalty (i.e., negative reward) in order to discourage such behavior in future runs. One main goal of this work is to design an action space that allows to effectively close the perception-motion loop. Since motion generation is highly depen-

dent on the perception system, our experiments are performed in a vision-based environment. We learn visual perception based on self-supervised representation learning. We first sample actions based on Brownian motion [188] to collect images from our environment. The ideal latent representation for motion generation should encode environment information such as object poses, and shape. More generally, it can contain any geometric information concerning the objects in the environment. Hence, for a self-supervised approach to capture that, the image data it is trained on should contain samples that represent a lot of variations of these variables. Therefore, during data collection, we use a smaller episode duration. Since obstacle types and poses are random at different episodes, having shorter episodes would result in more episodes for the same amount of data and hence more diverse obstacles and obstacle poses in the collected samples. Given a dataset of images, we use a (β-VAE) [73] to train a perception system that maps images $\mathbf{o}$ into latent representations $\mathbf{z}$. We use the β-scheduler proposed in [32] to improve the quality of the latent representation and its disentanglement while ensuring the final representation is based on the true evidence lower bound of the data likelihood.

7.3 EXPERIMENTS‡

We conduct experiments to answer the following questions:

(Q1) Is our method capable of learning successful obstacle avoidance strategies?

(Q2) How does the baseline policy contribute to the overall sample-efficiency?

(Q3) Is our approach capable of simultaneously avoiding multiple obstacles?

7.3.1 *Setup*

All of our experiments share the same setup. Namely, we use a seven DoF Franka Emika Panda robot in a Gazebo-based simulation environment [100]. We place a static RGB camera in front of the robot (green block in Figure 7.3). We use three different obstacle shapes: cuboids, spheres, and cylinders. Each obstacle has a distinct texture. In our experiments, we sample the object positions to be true obstacles with respect to the goal. The environment setup is illustrated in Figure 7.3. Furthermore, we use an NVIDIA GeForce RTX 2080 GPU to train our vision module, and reactive policy. For all of our experiments, we downsample the RGB images to 128×128. All of our results are averaged over 5 training trials with different random seeds. We look at the success rate and the average episode return (AER). For improved visibility, we smoothen the AER plots using running means

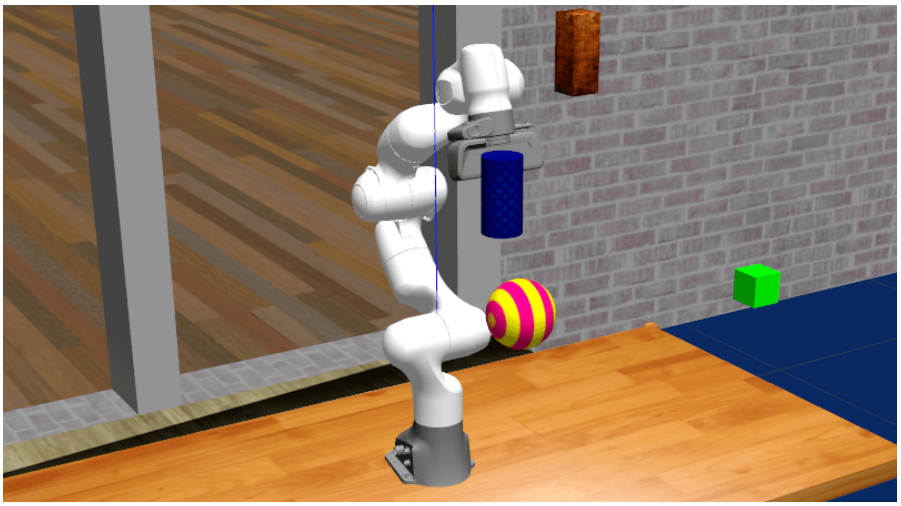


Figure 7.3: Illustration of our task setup in simulation. Single or multiple obstacles can be sampled in each environment instance. The robot is supposed to reach a given goal position without colliding with any obstacle. Obstacles can have distinct shapes, sizes and visual textures.
This figure has previously been introduced in one of the author's previous publications [4].

with a window size of 20 episodes. For measuring the success rate, we label a trial as successful, when the end-effector reaches the desired goal with no collision and manages to stay there for 5 seconds.

7.3.2 *Results*

Experiment A. In our first experiment, we want to evaluate the proposed method on a single obstacle avoidance task. Additionally, we want to see the effect of the baseline policy on the overall performance. We refer to the case without a baseline policy as **Vanilla Learning (VL)** and the one with baseline policy as **Residual Policy Learning (RPL)**. VL would then correspond to a standard Deep RL approach for obstacle avoidance with the addition of the latent model and the RMP handling of $\mathbf{a} = \mathbf{d}_q$. In Figure 7.4, we compare the average episode reward over time (left) and success rate (right) of both possibilities. After approximately 80 minutes, both policies converge and the average reward as well as the success rates remain more or less the same. The total reward accumulated using RPL is substantially higher than that of VL. The latter manages to avoid the obstacle in few trials, however, it fails in most cases to also reach the goal, or even get close to it. This behavior can be shown by looking at the individual reward terms from equation (7.5). We provide a plot of these values in Appendix B.2.3.1. Furthermore, vanilla learning requires at least twice the amount of samples to reach the same success rate as our method. This is mainly due to two factors. First, for our use case, VL attempts

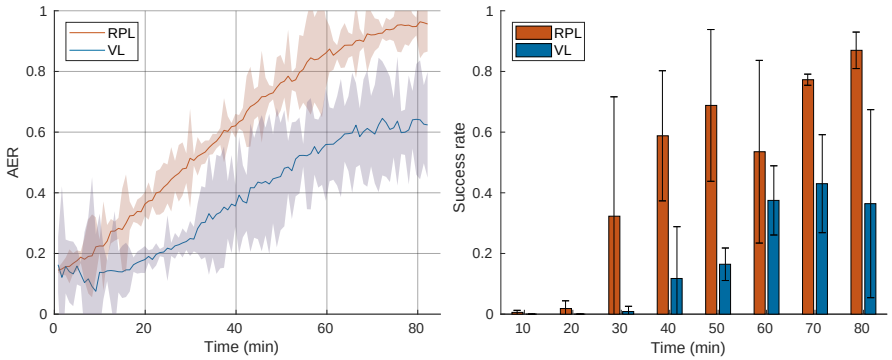
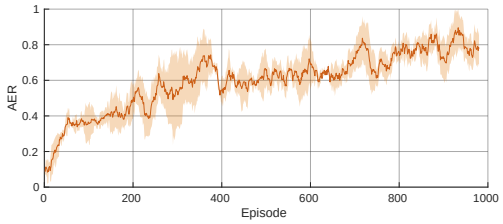


Figure 7.4: **Experiment A**. Effect of the baseline policy on the single obstacle avoidance performance. (Left) Normalized average episode reward (AER) over time. (Right) Success rate over time.
This figure has previously been introduced in one of the author's previous publications [4].

to learn both obstacle avoidance and goal reaching. However, in RPL, goal reaching is taken care of by the baseline policy. One could argue that providing a baseline policy limits the flexibility of the model, which usually leads to declined performance. However, our experiments show a different tendency. Even after convergence, our method achieves more than twice the success rate ($84 \pm 6\%$) than the vanilla method ($39 \pm 27\%$), which seems to often get stuck in a local solution (it reaches a lower average reward even after convergence). The second reason for this difference in sample-efficiency is exploration. In its early stages, our method already tries to reach the goal, as it is guided to do so by the baseline policy. While trying to do this, it also encounters the obstacles more frequently and receives negative reward. However, in the absence of a baseline, the exploration is random, and results mostly in low-reward trajectories for training.

Experiment B. In the second experiment, we aim to test the capability of our method to learn to avoid multiple obstacles simultaneously. The only difference is that we train the policy in an environment containing 3 obstacles at all times (as shown in Figure 7.3). As expected, the amount of interactions needed for this task to be learned is substantially higher than that of the single obstacle case. The policy converges after 10000 episodes, which is approximately equivalent to 3 hours of data collection with a single robot. The success rate after convergence is 72%. We show the evolution of the AER over time in Figure 7.5 (left). Furthermore, we test the obtained policy on environments containing one and two obstacles to check whether the learned reactive behavior can generalize to different scenarios or just memorizes sequences of actions depending on the obstacles configuration. We evaluate our policy for 50 trials per scenario. We report **success** as previously defined, **near goal** reaching which corresponds to avoiding the obstacle and getting to the close proximity of the goal (approximately 7cm) but not exactly reaching it, and **failure** which



# obstacles	Success	Near Goal	Failure
1	66%	30%	4%
2	54%	44%	2%
3	72%	22%	6%

Figure 7.5: **Experiment B** Multiple obstacle avoidance. (Left) Normalized average episode reward (AER). (Right) Generalization of a policy trained for 3 obstacles in environments with 1 and 2 obstacles. *This figure has previously been introduced in one of the author's previous publications [4].*

corresponds to all other cases. The results are shown in Figure 7.5 (right). Interestingly, even when only trained in 3 obstacles environments, the obtained policy still manages to succeed in reaching the goal or its near proximity at a high rate, even when tested on environments with 1 and 2 obstacles. These results strongly support our claim for generalization.

7.4 DISCUSSIONS

This chapter proposed an action space for handling the motion generation aspect of robotic manipulation tasks. The proposed action space is built on top of Riemannian motion policies to abstract certain basic aspects of motion generation from the policy. Mainly, goal-reaching, constraint satisfaction, and obstacle avoidance are formulated as RMPs. The RL policy is left with the task of inferring the task-space goal and in real time inferring configuration-space displacement vectors to avoid the obstacles. The task-space goal can in practice be the action of a high-level policy in a hierarchical RL framework. The low-level policy should then run at a higher frequency, and output the joint space displacement vectors for obstacle avoidance. We tested this approach in a simple reacher environment, which is why we could fix the goal in the current experiments and did not need the high-level policy. In this way, we could assess the benefits of this action representation in isolation from the hierarchical policy training process. Nevertheless, the experiments clearly show the advantages of factorizing the motion generation process as proposed. Learning a reaching policy using the proposed action space proved to be very sample-efficient. Experiments also showed that this action space can be used successfully in environments with multiple obstacles, which is a challenging problem in robotics, especially for high-DoF manipulator applications. These findings prove the usefulness of this action space and its suitability for manipulation tasks in unstructured and potentially cluttered environments.

However, as previously mentioned, motion is only one part of manipulation. The current action space does not explicitly include any mechanism for the policy to control the physical interaction of the robot with its environment. Of course, an RL policy can learn to abuse obstacle avoidance displacement vectors to push the low-level controllers to exert a force on the environment, but such a behavior can be unsafe and in some cases unfeasible, as described at the beginning of this chapter. The aspect of environmental interaction has already been explored in the literature on action representations, with publications suggesting the use of variable impedance control or force impedance control as action spaces for manipulation [126, 189]. These methods nicely complement our work and can be integrated into our framework to create an action space that properly handles both motion generation and physical interaction.

Finally, the main takeaway from this chapter is that the action space plays a very important role in learning robotic manipulation skills. Carefully building elaborate action spaces can simplify or even enable learning various skills. Future work should further explore this area and more efforts are needed to design good action spaces for manipulation. The recent trend in robot learning is to have one agent or at least one algorithm that is capable of simultaneously learning multiple tasks. This creates a need for action spaces that are suitable for multiple or perhaps all tasks. This work is one step towards this goal.

8

CENTRAL LATENT ACTION SPACES

The two previous chapters discussed methods for building action spaces suitable for manipulation. Such action spaces employ well-established control and motion generation methods as inductive biases. As a result, the complexity of the policy is reduced, and the training process should become more sample efficient as a consequence. However, the design of an action space typically lies on certain assumptions about what should be abstracted from the policy and what it should have full control over. These assumptions can be specified on the basis of the current knowledge of classical manipulation methods. It is possible to find assumptions that are common to most manipulation tasks, but this process is challenging and error-prone.

Alternatively, and similar to the state space, we can learn action representations for manipulation. The motive behind this approach is to eliminate designer bias from the action representations. In other words, the main goal is to reduce the effect of the assumptions made when designing the action space. Ideally, a learned action space could represent the actions in a lower-dimensional action manifold. It can also combine features from different control principles. For instance, an abstract latent action representation could represent variables that control both the joints of the robot and its end-effector. A latent action space could also be robot-agnostic, allowing policies to be used in different embodiments. More importantly, a learned action space could be made simultaneously suitable for multiple tasks without relying on assumptions of what is common against them. However, the paradigm of learning action representations introduces its own assumptions. The first assumption is that a latent action space useful for manipulation exists and has certain advantages over existing actions spaces. Another very important assumption is that there exists a valid mapping between the latent action space and an action space which the robot already supports, effectively allowing the latent command to be executed on the robot. Besides these assumptions, it is also important to note that the learned representations are as good as the data used to train the corresponding models. This is one of the main challenges of this paradigm. Despite the state and action space representing fundamentally different problems (perception and control, respectively), some methodology for learning their representations can in practice be shared. Therefore, we can draw inspiration from the more explored topic of state representation learning.

Unlike state representations, building action spaces remains more appealing than learning them. This is due to the fact that low-level

robot control is a long-standing problem with many well-established methods. As a result, most robot learning applications still rely on designed and not learned action representations with low-level controllers. This motivates the need for further research on learning action representations, but also the need to combine both paradigms.

8.1 LEARNING ACTION REPRESENTATIONS

In principle, we could use various representation learning algorithms to learn action representations. However, due to the need to be able to map latent actions into normal actions, we can only use methods that result in some generative process. In contrast, in state representation learning, we are mainly interested in the latent state inference process. The generative process is only needed during training by some methods like AE. This makes the usage of certain representation learning methods (such as contrastive learning) more difficult or even impossible.

Zhou, Bajracharya, and Held [219] proposed an approach to learn a latent action space using a conditional VAE. Their approach is designed for offline RL, where the latent action space is meant as a mechanism to inform the policy to output actions within the support of the dataset. Later a similar version of this work was proposed in [11]. This approach works in online RL and its main motivation is to learn a compact action manifold to simplify the policy search. Policy search is typically performed in the latent action representation. During inference, the decoder part of the AE is used to transform the latent action back into the original action space. Zhang et al. [218] extend the concept of latent actions to encode full action trajectories using transformers [194]. The main advantage of this approach is that it eases planning. However, it comes at the cost of reducing feedback in the overall system. Karamcheti et al. [90] propose an approach to learn language-informed latent action spaces using language embeddings from a pre-trained language models. A policy learned in this latent action space can later be influenced by human text commands to alter its behavior. Mehta, Parekh, and Losey [128] learn a latent action space that encodes different manipulation behaviors. The training data is collected autonomously based on an RL agent that learns to maximize the state entropy of the objects and the robot's interaction with the objects in the scene.

8.2 DECENTRALIZED COOPERATIVE CONTROL

One motivation for learning latent action spaces is to reduce the dimensionality of the action space used in policy training. In multi-robot manipulation tasks, the state and action spaces grow with the number of robots, making this problem even more challenging. As

a consequence, single-agent approaches seem unsuitable for these environments. Alternatively, in a multi-agent reinforcement learning (MARL) approach, each agent is responsible for actuating a part of the environment. Individual agents could have access to either all observations or just a subset. This simplifies the exploration and sample requirement for each individual agent. However, multi-agent methods suffer from the lack of information present to each agent, which results in multiple problems [37, 60]. Most notably, from the perspective of each agent, the environment is no longer stationary, as the other agents are now part of the environment and regularly update their policies and behavior [60]. This increases the difficulty of policy search. Furthermore, the lack of information about the other agents' actions makes coordination and estimation of interaction dynamics even harder than usual. Hence, in the literature, multiple solutions have been proposed to approach these problems.

Many MARL methods attempt to establish either an explicit communication channels such as in [44, 61, 137, 151, 172, 178], or an implicit information exchange as part of the policy architecture or the learning algorithm [62, 109, 120]. For instance, multiple methods are based on modeling the other agents' policies [118, 154, 213]. This, however, comes with the burden of training and tuning $N \times N$ policies for N agents. This kind of method is usually based on the centralized training decentralized execution (CTDE) paradigm, where at training time, each policy can benefit from the information that is usually exclusive to the other agents at execution time. Others have proposed using CTDE to learn a central dynamics model and use the model to train decentralized policies [203, 217]. Similarly, Lowe et al. [120] and Foerster et al. [53] propose training decentralized actors using a centralized critic. Another common approach is to decompose the value function to the different agents [156, 179]. Beyond CTDE, multiple other solutions have been proposed to alleviate the non-stationarity of MARL tasks. For instance, Liu et al. [117] propose engineering the reward function to punish competitive actions taken by individual agents. Others proposed designing learning curricula that enforce a similar tendency [31, 133]. Gupta, Egorov, and Kochenderfer [62] relied on policy parameter sharing across agents, which allows multiple agents to use the same policy network while passing an agent index as part of the observation. Furthermore, MARL solutions could be simplified using custom policy parameterizations such as finite state controllers [12, 22] or transforming the problem to enable tractable planning and search [47, 48]. However, decentralized MARL methods fail to achieve the level of coordination, which is necessary for the control of physical systems. A more extensive overview of MARL methods can be found in [216].

Another challenge for multi-agent systems is the decentralized action generation. This problem could also occur when each agent has access to the full state, but is only actuating a part of the overall sys-

tem. This aspect can be ignored for the classical application domains examined by previous MARL research, such as games and particle environments. However, it becomes critical when dealing with physical tasks, such as dual-arm manipulation, where decoupled actions could lead to instabilities and even damage the robots. Lee, Yang, and Lim [109] tackle this problem by first learning robot-specific skills and then learning a meta-policy that selects the skills each agent should execute.

Decentralized cooperative control tasks can be studied as decentralized partially-observable Markov decision processes. The latter is a special type of partially observed stochastic games. It is defined by the set $\langle N, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\{1,\dots,N\}}, \mathcal{P}, \{r_i\}_{i\in\{1,\dots,N\}}, \gamma, \{\mathcal{O}_i\}_{i\in\{1,\dots,N\}}, \rho\rangle$, where N is number of agents ($N = 1$ corresponds to the single-agent problem), $\mathcal{S}$ is the state space across all agents, $\mathcal{A}_i$ is the action space for the i^{th} agent, $\mathcal{P}$ represents the environment dynamics, r_i is the reward function for the i^{th} agent, γ is the discount factor, $\mathcal{O}_i$ is the observation space of the i^{th} agent, and ρ is the initial state distribution. In cooperative tasks, all agents share the same reward $r_1 = r_2 = \cdots = r_N$. Optimally solving partially-observable Markov decision processes is a challenging combinatorial problem that is NEXP-complete [23] in contrast to MDPs, which are P-complete [144].

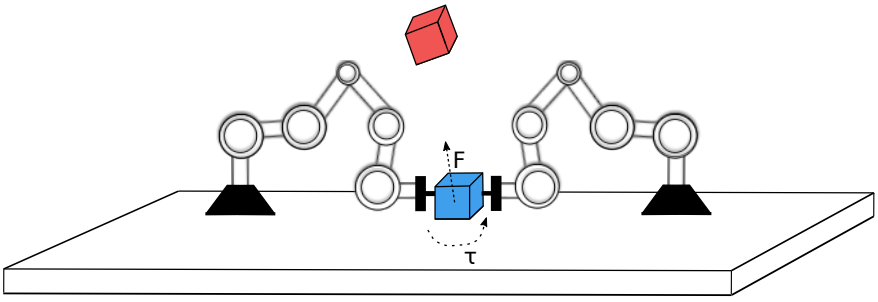


Figure 8.1: Two robot arms cooperating on an object lifting task. The red cube indicates the target pose. Traditionally, two agents would control the separate robot arms in a control space of the robot such as joint torque control. We explore the option of learning latent central actions spaces which are robot-agnostic and central to the task. In our example, a possible action space would correspond to the force F and torque τ acting on the center of mass of the cube.
This figure has previously been introduced in one of the author's previous publications [7].

8.3 CENTRAL LATENT ACTION SPACES FOR MULTI-ROBOT MANIPULATION

Single and multi-agent approaches to multi-robot manipulation define a different action space for each robot. However, for many MARL

tasks and especially for multi-robot manipulation, the task is defined with respect to certain task variables such as the position of an object. Let's consider a simple example of dual-arm lifting as shown in Figure 8.1. In this task, two robots cooperate to lift an arbitary object to a defined goal pose. Independent of the choice of method, performing this task would require commanding each robot in a way that fulfills the task goals. Control can be framed in the task or configuration space and using either position, velocity, acceleration, or force/torque targets. However, what matters to fulfill this task is the trajectory of the object itself. This trajectory can be characterized using the object's position, velocity, acceleration, or the forces and torques acting on the object. Controlling one of these variables can successfully lead to fulfilling the task. If we ignore the robots for a second and assume that we have an environment where we could directly control one of these variables, we can then define an action space that is acting directly on the object. An ideal example would be to define the action space to be a force and torque acting on the center of mass of the object. Training an agent to perform actions in this space could successfully solve the task. More importantly, this action space does not grow with the number of robots since it assumes actions on the object directly. Any other action space that is robot-agnostic would fulfill these same properties, even if it does not have a physical meaning or if it represents a purely semantic entity.

However, if we have an agent that is capable of controlling the object in isolation, it is not straightforward to map these actions into control commands that the robot can execute in a way to cause these object actions to happen in a real embodied world. In other words, it is very difficult to compute joint torques (or any other control) that each robot needs to apply to cause a given force (or any other control) to act on the object. This becomes even harder if the action acting on the object does not have a strictly physically interpretable meaning. In this work, we postulate that there exists a latent action space that resembles the one just described, and we aim to learn a generative model that can generate actions in the original control space of the robot based on actions in this new space. Unlike previous work on learning action representations, we require the generative process to be decentralized to still allow for each robot to have its own policy and autonomy. The main motivation behind this is that such a central action space does not grow with the number of robots. If we can still generate a raw action for each robot using this latent action, we would keep all the nice properties of decentralized agents while achieving coordination based on the central action.

To learn a latent central action space, we use Stochastic Gradient Variational Bayes [98] to overcome the intractable inference distributions involved in learning mappings to this space. First, we look at the case where each agent receives full observations $\mathbf{o} \in \mathcal{O}_1 \times \mathcal{O}_2 \times \cdots \times \mathcal{O}_N$. For that, we introduce the models in Figure 8.2 (left). The

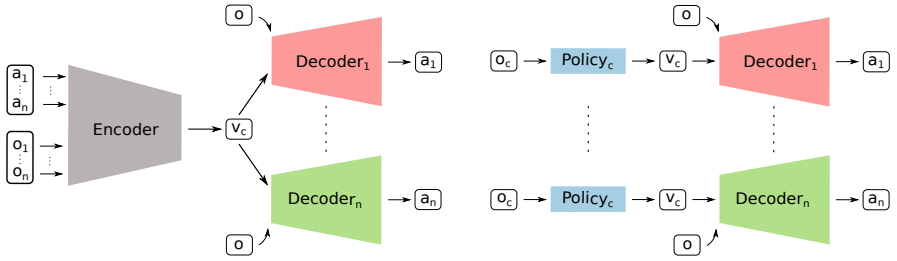


Figure 8.2: System overview under full agent observability. (Left) we use a conditional autoencoder for learning the central latent action space. The encoder receives all observations and actions from all agents and produces a latent action $\mathbf{v}_c$. This latent action together with the full observation is given to the agent-specific decoders together with the observation. Each decoder outputs an action that is in the original action space of the corresponding agent. (Right) All agents share the same policy acting in the latent action space. The learned decoders map the latent action into the original action space.
This figure has previously been introduced in one of the author's previous publications [7].

generative process (encoding) of each agent's original action $\mathbf{a}_i$ is conditioned on the latent central action $\mathbf{v}$ and the observation $\mathbf{o}$. The latter is also used during the inference process (decoding), as shown in Figure 8.2 (left). Additionally, we encode latent actions based on the actions from all agents $\mathbf{a} = [\mathbf{a}_1, \dots, \mathbf{a}_N]$. This is possible since the inference/encoder network will not be used at execution time, and will instead be replaced by the policy as shown in Figure 8.2 (right). Based on this model, all agents could share a copy of the same policy, which outputs a latent central action $\mathbf{v}$ based on the full observation $\mathbf{o}$. However, they would each have a different decoder to translate the latent action $\mathbf{v}$ into their original action space. This is illustrated in Figure 8.2 (right). Having a shared policy is feasible in this scenario since the latent action space is supposed to have a lower dimensionality than the aggregated action space of all control agents. To illustrate this, we go back to the lifting example. Controlling the joint velocity of two robots with six DoF would result in an action space with a dimension of twelve. Instead, controlling the wrench applied to the object only requires an action space with six dimensions. Note that this number does not grow dramatically with the number of agents or robots. We derive a lower bound to the marginal likelihood

$$
\begin{aligned}
p(\mathbf{a} \mid \mathbf{o}) &= \int p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v})\, p_\psi(\mathbf{v} \mid \mathbf{o})\, d\mathbf{v} \\
\ln p(\mathbf{a} \mid \mathbf{o}) &= \ln \int p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v})\, p_\psi(\mathbf{v} \mid \mathbf{o})\, \frac{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a})}{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a})}\, d\mathbf{v} \\
&\geqslant \int q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a}) \ln(p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v})) \frac{p_\psi(\mathbf{v} \mid \mathbf{o})}{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a})} d\mathbf{v}
\end{aligned}
$$

$$= \mathbb{E}_{q_\phi(\mathbf{v}|\mathbf{o},\mathbf{a})}[\ln p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v})] - \mathrm{KL}\big(q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a}) \,\big|\big|\, p_\psi(\mathbf{v} \mid \mathbf{o})\big) \tag{8.1}$$

$$= \mathcal{L}(\mathbf{a}, \theta, \phi, \psi \mid \mathbf{o}), \tag{8.2}$$

where KL(||) is the Kullback-Leibler divergence, $q(\mathbf{v} \mid \mathbf{o}, \mathbf{a})$ is the approximate posterior distribution

$$q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{a}) = \mathcal{N}\big(\mathbf{v}; \mu_v, \sigma_v^2\big), \quad [\mu_v, \sigma_v] = g_\phi(\mathbf{o}, \mathbf{a}). \tag{8.3}$$

Since the generative process of each agent's action is distributed, the likelihood is composed of multiple terms.

$$p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v}) = [p_{\theta_1}(\mathbf{a}_1 \mid \mathbf{o}, \mathbf{v}), \ldots, p_{\theta_N}(\mathbf{a}_N \mid \mathbf{o}, \mathbf{v})], \tag{8.4}$$

where θ_i refers to decoder parameters for agent i, and $\theta = \{\theta_i\}_{i \in N}$. Note that the prior is conditioned on the observations. It is parameterized by ψ and has a policy-like form $p_\psi(\mathbf{v} \mid \mathbf{o})$. We train it simultaneously to the encoder and decoders using the same loss function from equation (8.2).

As previously mentioned, agents in a partially-observable Markov decision processes have only access to a subset of the observations. However, we notice that in most environments, a certain part of the observations is shared across all agents, and that is usually related to either objects in the scene or any kind of other task-specific observations; but not to the agent's embodiment. Even when this condition fails, it could be enforced in the learning process. For instance, in [119], the latent space is designed to contain information about the object relevant to the task.

We introduce a new set of models, as seen in Figure 8.3 (left). In this new model, the latent action space is partitioned into $N + 1$ parts. The first N correspond to latent actions $\mathbf{v}_i$, which are specific to each agent. The last part $\mathbf{v}_c$ is central and shared with all agents. The generative process of each agent's action (in the original action space) is now conditioned on the agent's observation $\mathbf{o}_i$, the latent agent-specific action $\mathbf{v}_i$, and the latent central action $\mathbf{v}_c$. As for inference, the whole latent action variable is conditioned on the full observation $\mathbf{o}$ and the full action $\mathbf{a}$. As in the previous case, using the full observation and action for inference is possible because the encoder would not be used during control. Instead, each agent has a set of two policies: one policy producing the latent agent-specific action $\mathbf{v}_i$ based on $\mathbf{o}_i$; and another policy that is shared across all agents, and which generates the latent central action based on the shared observation $\mathbf{o}_c$. These two latent actions are then concatenated and decoded into the original action space of the agent. We show the architecture of the policy in Figure 8.3 (right). Note that the policy updates also

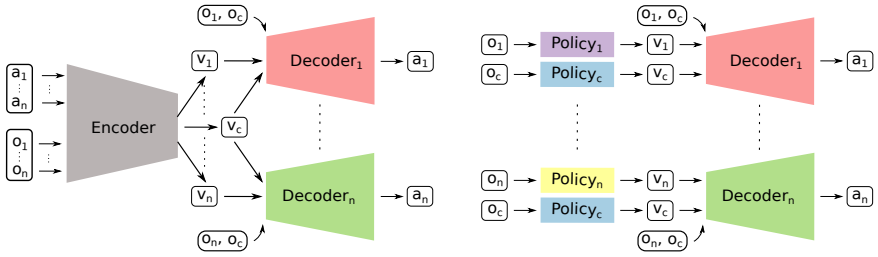


Figure 8.3: System overview under partial agent observability. (Left) We use a conditional autoencoder to learn the central latent action space. The encoder receives all the observations and actions of all agents and produces a latent action $\mathbf{v}$. The latent action contains agent-specific actions $\mathbf{v}_i$ as well as a central latent action $\mathbf{v}_c$. This latent action together with each agent's observation are given to the agent-specific decoders together with the observation. Each decoder outputs an action that is in the original action space of the corresponding agent. (Right) All agents share the same policy acting on the object in the latent action space. Each has a separate policy acting in the latent agent-specific action space. We use the learned decoders to map the latent action into its original action space(s).
This figure has previously been introduced in one of the author's previous publications [7].

affect the decoder. The new lower bound is very similar to the one in equation (8.2), with the minor difference of

$$p_\theta(\mathbf{a} \mid \mathbf{o}, \mathbf{v}) = [p_{\theta_1}(\mathbf{a}_1 \mid \mathbf{o}_1, \mathbf{o}_c, \mathbf{v}_1, \mathbf{v}_c), \ldots, p_{\theta_N}(\mathbf{a}_N \mid \mathbf{o}_N, \mathbf{o}_c, \mathbf{v}_N, \mathbf{v}_c)]. \tag{8.5}$$

The encoders, decoders, prior distributions, and policies involved in this method are implemented as multi-layer neural networks using PyTorch [146]. All distributions are transformed Gaussian distributions using a hyperbolic tangent function ($\tanh$). Actor, critic, and prior networks have two hidden layers. Encoders and decoders have three hidden layers. For training, we use the Adam optimizer [97]. Each policy is optimized using soft-actor-critic (SAC) [64]. All modules are trained using randomly sampled data from the replay buffer. The latter contains trajectories sampled from the previously described multi-agent policy. At the beginning of training, we only update the latent action model using random actions in a warm-up phase that lasts for a hundred thousand steps. We found this step to help the training performance and stability. For a fair comparison, we also implement this step for all the considered baselines.

8.4 EXPERIMENTS‡

We designed our experiments to investigate the following questions:

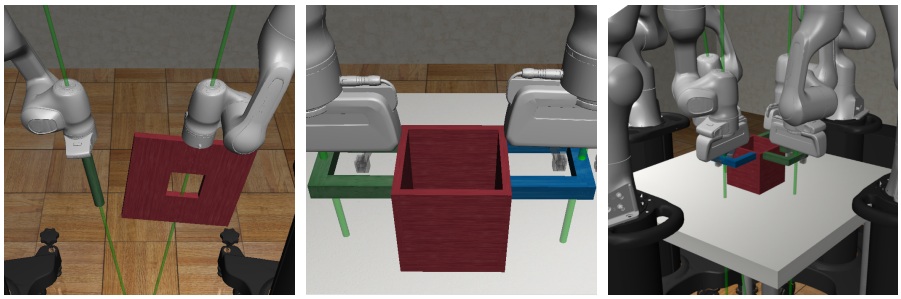


Figure 8.4: Close-up screenshots from the simulation environments used in our experiments. The environments are provided by robosuite [220]. (left) dual-arm-peg-in-hole environment (middle) dual-arm-lift and (right) four-arm-lift environment with modified gripper structure.
This figure has previously been introduced in one of the author's previous publications [7].

(Q1) Can central latent action spaces help coordinate action generation in decentralized cooperative robotic manipulation?

(Q2) Does our method improve sample efficiency with respect to the selected baselines?

(Q3) Can our method reach or exceed the performance of single-agent approaches with full-state information?

(Q4) Is our method scalable to more than two-arm manipulation tasks?

(Q5) How robust is our method to external disturbances?

8.4.1 *Environments*

We evaluated our method in three simulated environments based on robosuite [220]. The environments are selected/built such that they require cooperation between multiple robot arms. Due to the lack of standardized environments that are suitable for our use case (i. e.multi-robot manipulation), we use existing environments from public benchmarks when suitable and build alternatives when needed. Due to the nature of our problem, we select environments that have continuous state and action spaces. In all of the environments, each agent's observations are the corresponding robot's joint position and velocity, as well as its end-effector pose.

DUAL-ARM-PEG-IN-HOLE. In the first environment, 2 robot arms cooperate in the peg-in-hole task. A close-up view of the scene and the objects can be seen on the left in Figure 8.4. We are using the original reward from robosuite, which is composed of a reaching and orientation reward. The shared observation $\mathbf{o}_c$ corresponds to the poses of the peg and hole and the distance between them.

DUAL-ARM-LIFT. For the second environment, we decided to use the dual-arm-lift environment. In this environment, a rectangular pot with two handles is placed on a surface between two robot arms. The task for each robot is to reach and grip the handle before cooperatively lifting the pot off the surface. During initial experiments, we noticed that the provided reward does not promote cooperation and can be easily tricked, i.e., the maximum reward per time step can be reached by controlling a single robot to tilt and lift the pot only slightly off the table. This is due to the generous maximum tilt angle of 30° and the successful lift height of 0.10. The other major component of the reward measures the ability to reach and grasp the pot handles. However, we are not interested in assessing the reaching and gripping capabilities but want to rather reward cooperative lifting behavior. Therefore we are considering the following modifications to the reward of the environment. At the start of an episode, we move each robot's end-effector close to its handle and weld the pot handle to the end-effector with a distance constraint in the MuJoCo ([187]) simulator. We chose a distance constraint because it constrains the position but leaves rotational coordinates free. We remove the gripper fingers to avoid unwanted collisions. We visualize the resulting starting condition in the middle of Figure 8.4. We also modify the reward function to enforce success only during high lifts. Additionally, the maximum tilt angle is reduced such that both robots must cooperate to keep the pot level at all times. The shared observation $\mathbf{o}_c$ corresponds to the pose of the pot.

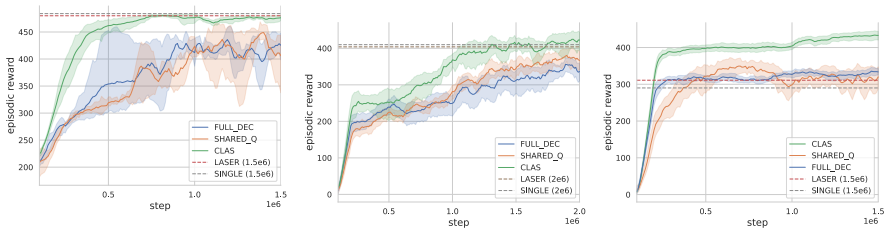
FOUR-ARM-LIFT. The third environment is an extension of the dual-arm-lift environment and uses two additional robot arms to lift the pot (i. e.total of four robot arms). Here the pot weight is increased to keep the coordination requirement. We build this environment for the sole purpose of testing the scalability to more than two robots/agents. The pot with four handles and the robot arms' placement can be seen on the right in Figure 8.4.

The changes to the lifting environments were evaluated with manual human control to ensure that tricking the system or solving the task with a single robot arm is not possible. Keeping a high reward was only possible when the pot is lifted vertically for a long period of steps. All environments use a joint velocity controller which receives desired joint velocities from the policy.

8.4.2 *Baselines:*

To validate our method, we compare it to well-established baselines that have been previously applied to continuous control. Our experiments include the following baselines:

- `SINGLE`: refers to having a single agent controlling all robots.



(a) dual-arm-peg-in-hole (b) dual-arm-lift (c) four-arm-lift

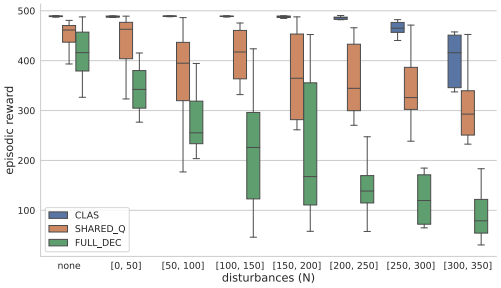
Figure 8.5: Performance in simulated multi-robot manipulation tasks under Partial-observability. We compare our method (CLAS) to centralized single-agent and decentralized multi-agent approaches. Our approach outperforms the considered decentralized multi-agent approaches in all environments. It also manages to solve the four-arm-lift task in which all the considered baselines fail.
This figure has previously been introduced in one of the author's previous publications [7].

- `LASER`: uses a latent action space on top of a single agent controlling all robots. This is based on the work in [11].
- `FULL_DEC`: refers to having all agents trained with the exact observations and actions they will have access to during execution. The agents are not provided with a communication channel.
- `SHARED_Q`: similar architecture to `FULL_DEC`, but all agents are trained using a central critic. This baseline is based on the work in [120].
- `CLAS`: refers to our method and abbreviates "central latent action spaces".

The first two single-agent approaches are included as strong baselines and references. They serve us to better understand the different environments and to elaborately analyze our results. Finally, to make the comparison more reliable, we use SAC for training the different agents in all baselines.

8.4.3 *Results*

Task Performance. Figure 8.5 shows the episodic reward obtained by our method and the baselines in the three environments considered. Looking at the single-agent approaches, we observe that both baselines reach high-reward areas for the dual-arm tasks. However, both fail to solve the four-arm-lift task. At the end of training, the best mean episode reward achieved by a single agent is substantially smaller than the maximum possible reward and has a very large variance. This illustrates the problem of learning multi-robot manipulation tasks with large action and observation spaces with a single-agent RL approach. In contrast to the dual-arm tasks, the four-arm-lift environment features state and action spaces twice the size. Next, we analyze the results from MARL-based methods. `FULL_DEC`



Success rates over 10 evaluation runs under external disturbances.

	FULL_DEC	SHARED_Q	CLAS
none	70%	**100%**	**100%**
[0, 50]	30%	80%	**100%**
[50, 100]	20%	60%	**100%**
[100, 150]	10%	70%	**100%**
[150, 200]	20%	40%	**100%**
[200, 250]	0%	40%	**100%**
[250, 300]	0%	30%	**90%**
[300, 350]	0%	20%	**60%**

Figure 8.6: Effect of applying disturbances (forces) at the center of mass of the pot in the four-arm-lift environment. (left) Episode reward (right) success rate under different ranges of disturbances. The results are based on 10 evaluation runs. Our method demonstrates robustness against different ranges of disturbances in comparison to the other decentralized baselines, which success rate decreases dramatically as we increase the disturbance.
This figure has previously been introduced in one of the author's previous publications [7].

and SHARED_Q struggle to keep up with single-agent RL methods. Both methods do not explicitly encourage coordination. Hence, this result might indicate that our environments are well-suited for studying partially-observable Markov decision processes, since they require a certain degree of coordination to be solved. The two approaches manage to solve the peg-in-hole task but struggle in the two other environments. They also lead to very similar results. In contrast, our method (CLAS) successfully solves all tasks even under partial observability. In the dual-arm-peg-in-hole environment, it reaches a high episode reward after only 250 thousand environment interaction steps, while the two other MARL approaches fail to do so in triple the number of steps. Furthermore, it achieves a final performance very close to the one achieved by single-agent methods. In the dual-arm-lift environment, our approach outperforms both MARL-based baselines. Additionally, it surpasses the final performance of the two other MARL approaches after only half the amount of steps. More importantly, CLAS slightly outperforms the single-agent methods. In the four-arm-lift environment, CLAS is the only studied method that manages to solve the task and achieve a high reward. Even the single-agent baselines which have access to full state information fail in this task. This indicates that acting in the latent central action space enables coordinated control even under partial observability and action decentralization. Finally, we notice that our method leads to significantly lower performance variance, which makes deploying it in real-world scenarios more reliable.

Robustness analysis. We aim to evaluate the coordination capability of our method by quantifying its robustness to external disturbances. We perform this experiment on the four-arm-lift environment and compare the different decentralized baselines to our method. For each method, we pick the model from the training run with the best-achieved performance. We then evaluate the corresponding agents in the same environment as before, however, when additionally applying an external force to the pot. Force is applied during the steps in the interval [10, 100], and force vector values are uniformly sampled at each step to be in a certain range. We experimented with multiple ranges. The results can be seen in Figure 8.6. Under no disturbances ("none"), all methods achieve a high reward and a decent success rate. After applying disturbances in the range [200-250], `FULL_DEC` fails in all evaluation runs to solve the task. The success rate of `SHARED_Q` goes down to 40%, but its reward remains relatively high as the agent manages to lift the pot a bit, but not always to the target height. On the other hand, our method `CLAS` is almost not affected by this level of disturbance. As expected, when increasing the magnitude of the forces, all methods start to fail more often at solving the task, but `CLAS` appears to remain reasonably robust.

We perform additional ablations and experiments to validate the coordination behavior of our method, and study its performance under full agent observability and asymmetric action spaces with different robots per agent. We also attempt to interpret the learned actions spaces. The results can be seen in Appendix B.3.

8.5 DISCUSSIONS

This chapter introduced a method for learning central latent action spaces for coordinating multi-robot manipulation tasks. The proposed approach simultaneously addresses two important problems in learning multi-robot manipulation. The first problem is that the action space grows with the number of robots, making exploration and policy training very expensive and in many cases not feasible. The second problem is the lack of coordination in decentralized multi-agent approaches. The proposed method combines good properties from single-agent methods (through the central latent action space) and decentralized multi-agent methods through the decentralized generation of raw actions. The experiments demonstrate the suitability of this method in various multi-robot manipulation tasks and its superiority over a selection of single-agent and multi-agent baselines. The scalability of the method to multiple robots and its robustness to external disturbances are clearly supported by the experiments. More importantly, the proposed method enables learning a four-arm lifting task that all considered baselines fail to solve. This shows that the method strikes a good balance between multi-agent coordination and sample efficiency.

One main disadvantage of this method is that it encourages policies to be task-specific. This reduces the usability of this method in multi-task domains. However, the concept of central latent action spaces and the proposed training method can be used for different multi-task purposes. For instance, future work could explore learning central latent action spaces that encode the constraints of the physical interaction between the different robots. Such an action space can be used with either multi-task or single-task data and later be used in any of those domains. Learning policies in such an action space would then ensure that these constraints are not violated. The central latent action space could also encode different properties that are not task-specific. For instance, for multiple tasks, an action that defines the distance between the different end-effectors and a given object can be sufficient for solving the task. Both ideas can theoretically work with minor adaptations to the current method. Another important limitation of this work is that it does not strictly enforce a physically meaningful action to emerge. This can be seen as a disadvantage since policies typically output interpretable actions directly usable with the environment's robots. In the proposed method, the policies actions can only be interpreted after they get decoded into the original action space. Future work can explore the imposing of physical constraints on the latent action space and ensuring that it is interpretable. This can be done, for instance, by introducing a latent dynamics model that follows the structure of rigid body dynamics equations.

Both disadvantages are common to all learned action representations. Recently, large-scale open-source datasets with data spanning multiple robots and multiple action spaces started to emerge [142]. This is very promising for learning task-agnostic and physically interpretable latent action spaces. The latter can be very useful for reducing the requirements for robot learning research and potentially enable new tasks to be solved. As mentioned above, a very promising direction for latent action spaces is to encode embodiement constraints. In addition to sample efficiency and exploration, such action spaces could improve the safety of RL methods. This can be true both for the execution and training phases. The latter being very important since it might simplify collecting real-world robotics data to train future models and policies.

Part IV

CONCLUSIONS

9

DISCUSSIONS

The motivation behind this thesis was to improve and build on top of state-of-the-art learning-based robotics manipulation. The aim was to bring the field of robot learning at least one step closer to making a real-world impact. However, the topic is very complex and has been researched for a very long time now, while still leaving much room for scientific curiosity and improvements. Its theoretical complexity stems from the inter-disciplinary nature of the problem being situated at the intersection of robotics, control theory, machine learning, and computer vision. In practice, the field moves slower than other ML-related topics due to multiple technical challenges. One major hurdle is data acquisition. While it is possible and rather easy to crawl the Internet for image and textual data, physical interaction is not really available in that context. In contrast, collecting robotics data can be very challenging and expensive and requires well-maintained software-hardware infrastructures. Recent progress in simulation technology [124, 187] alleviates some of those data challenges. However, even if we ignore the sim-to-real gap (which we should not), simulation environments are typically not readily available for each new task and instead require considerable engineering efforts. Recent work has also benefited from teleoperation and kinesthetic teaching data for training control policies via supervised imitation learning [125, 142, 168, 170]. However, such data is a lot harder to obtain than simply browsing the internet. Finally, unlike other fields in ML, robot learning still lacks a common real-world benchmark. This has meant that different laboratories can use completely different hardware setups and software implementations, making any comparison of results biased or in some cases unfeasible. However, this lack of benchmarks is also understandable, since robotic hardware quickly evolves and the field has not yet converged to a single platform that fulfills all research requirements.

Throughout this thesis, the field of robot learning has seen multiple trends, paradigm shifts, and changes in narrative. Some of the contributions in this thesis (especially the earlier ones) were influenced or inspired by these changes. Hence, before concluding the thesis and giving an outlook for future work, we will briefly revise and describe some of the major paradigm shifts and trends. The aim of this overview is to provide context and better place the contributions of this thesis.

9.1 PARADIGM SHIFTS AND TRENDS

One big decision required for designing a robot learning pipeline concerns the definition of what exactly is to be learned and how to learn it. Let us first assume that the answer to what is "everything" and how is in question. Early works in deep RL advocated for end-to-end learning of robotics policies [111, 196]. Under this paradigm, all aspects of the control policy such as perception, planning, and control should be learned with the same method and the same objective function. Whether through RL or imitation learning, all these components would be trained simultaneously. The advantage of such an approach is that all of these components would have some kind of mutual understanding of one another. For example, if the perception parts of the policy are erroneous, the downstream modules would know how to compensate for that, since they were trained together. This might at least be true under the training data distribution. Another advantage is that the earlier modules such as perception and planning would be targeted to the task at hand, since they are all trained with data from this task and potentially an objective related to it. The latter might only be true for RL-based methods.

However, being task-specific can become a disadvantage when considering more general robots. Additionally, learning all components using only an RL objective can be very challenging and expensive. Maximizing task reward does not directly provide any signals for learning perception. This might then lead the policy to learn some shortcuts or heuristics for control from high-level observations. This challenge led to the emergence of a different paradigm, where different parts of the pipeline can be learned separately, with different objectives, potentially different data, and even at different times. These methods relied mainly on SRL for learning perception, sequential models for planning [26, 45, 204] and learned action spaces for control [7, 11, 219]. Besides the gains in sample efficiency, an advantage of this paradigm is that different modules can be more easily reused for new tasks.

Another paradigm shift was related to the question of what should be learned. As previously hinted, early deep RL methods favored learning as much as possible [111, 215]. However, recent methods have seen an increased usage of non-learned components in the decision-making pipeline. These methods assume access to perception, planning, or skill library modules [34, 80, 115, 123, 195, 206]. However, it is unclear whether this tendency is indeed a paradigm shift, or just a simplification needed for making progress with large foundation models for robotics [30, 50].

Finally, in recent years, there has been a change from RL directly in the real world, to either RL in simulation combined with sim-to-real transfer [3, 38, 69, 183, 184], or imitation learning based on real-world demonstrations [125, 142, 168, 170]. This change was probably

sparked by the difficulty and expense of training data-hungry deep RL methods on expensive robots. However, while sim-to-real transfer and imitation learning alleviate the data problem, they do not offer a direct alternative to RL in the long term. This is simply due to the inability of these methods to directly query new data in the real world environment to overcome the distribution shift they might encounter at deployment time.

This is to hint, that some of these changes might be there to stay and indeed be classified as paradigm shifts, while others might be only short-term solutions to enable progress in the field. Robot learning is a very complex subject to a point where finding one common solution might not be possible like in other ML-centric fields. Instead, different methods, paradigms, and combinations are needed for different tasks. Even if a common solution does exist, it might be a combination of already existing paradigms or completely different ones. This thesis experimented with different paradigms and combinations in the search for a better understanding of the problem, and perhaps of the optimal approach (if it exists).

9.2 CONCLUSION

This thesis approached robot learning from the perspective of state and action space representations. The main contributions were split between the state and action aspects of the problem. Thus, it proposed ideas and concepts at the intersection of robotics, control, machine learning, and computer vision. The first part of the thesis deals with perception, and the state space. It first proposed a method for building suitable representations based on supervised object tracking. The proposed approach was tailored for robotics as it aimed to improve the tracking robustness in the presence of appearance changes. In addition, this thesis introduced a method for improving agent exploration in a way that targets state representation learning for model-free RL, or more generally, in the absence of a sequence model of the dynamics. The proposed method leverages different SRL objectives to train policies that can explore areas of the state-action space where the perception module struggles.

The second part was focused on the action and control aspects of robot learning. Unlike the state space, the action space perspective of robot learning received way less interest and research from the research community. Hence, to provide a better understanding of this problem, we first performed a study to understand the role of the action space in robotic manipulation learning. The study analyzes the role of different action space characteristics in exploration, task performance, and sim-to-real transfer of manipulation policies. As a result, the study provides explanations for the role of different design components in manipulation learning. Furthermore, it provides recommendations for research on building control spaces and appli-

cations in robotic manipulation. Similar to the state space part of the thesis, this part also experiments with building and learning representations, in this case, for the action space. Since manipulation is concerned with motion and interaction, we first propose a method for learning motion-centric action spaces that embed principles from differential geometry and motion primitives as inductive biases. The proposed space simplifies motion generation by abstracting some of its components from the policy. This abstraction reduces the task complexity from the policy's perspective. As a result, learning collision-free reaching skills can be done in a very short time. In addition, the thesis proposes an approach for learning latent action spaces based on variational inference. The method is tailored for multi-robot manipulation problems. It assumes the existence of a central latent action space which dimensionality does not grow with the number of robots. The method then learns inference and generative models that enable learning policies to act in this latent action space, while mapping these latent actions back into the robot's native control space in a decentralized fashion. This method enables scaling robot learning to multi-robot scenarios in an efficient manner.

In summary, the thesis tackled multiple challenges in the field by building and learning state and action representations. While sample efficiency is the common objective among all individual contributions, some also improved the exploration, robustness, and scalability of the learned controllers.

10 FUTURE WORK

This thesis proposed methods for obtaining state and action representations for learning manipulation. State representations have attracted a lot more research interest than actions. The ideas proposed here are only steps towards obtaining better action spaces. Future work should focus on finding more generally applicable action spaces that enable efficient manipulation learning. Ideally, these future action spaces would strike a better balance between complexity abstraction and policy flexibility. Classical robotics and control theory offer a lot of ideas for building such action spaces. Furthermore, learning latent action representation is also a very promising direction. Future work could investigate the efficacy of more recent generative ML methods such as transformers on this task. Additionally, it would be interesting to enforce constraints on these latent spaces to ensure some behavioral properties, such as smoothness or compliance. Another interesting avenue of research is on learning object-centric action representations, potentially using differentiable implementations of rigid body dynamics.

While the state and action space are very important aspects of the robot learning problem, they only constitute a small part of it. For example, looking back at the MDP definition, we immediately notice that two major components are not considered in this work. That is, the reward and dynamics models can play a major role in improving the efficiency and usability of robot learning. For dynamics, one could learn a sequential model and potentially integrate the state and action space representation objectives in the learning process. Although the latter is, to our knowledge, not well studied, integrating state space models in model-based RL is a well-researched topic [67, 91]. With the success of state-space sequential models, action space representation in sequential models seem like a promising avenue for future work. Such models could simplify learning dynamics, for instance, by abstracting complex dynamics with higher-level actions. However, the complexity of real-world dynamics stood in the way of using model-based RL in robotic manipulation, making such action abstractions even more applicable and promising in this domain. Furthermore, learning reward models can help overcome the reward shaping problem and alleviate some of the challenges in sparse rewards environments.

Beyond the MDP definition, robot learning still faces multiple challenges. For instance, scaling state-of-the-art methods to multi-robot tasks can be very challenging, as discussed in Chapter 8. The same is true for multi-task learning and more generalist approaches. Hi-

erarchical policies offer a promising framework for addressing some of these challenges, particularly when incorporating diverse sources of information. Additionally, embodied general agents could benefit from combining multiple levels of information such as semantic, affordance and physical knowledge. Recent work has shown that we can leverage high-level semantic knowledge in foundation models for high-level robot task planning [34, 80, 206]. These models can comprehend and generate human-like language, enabling the robot to understand complex instructions and abstract concepts.

To enhance mid-level planning, affordances can be integrated into the hierarchical policies. Affordances provide information about the interactions between objects and the environment, guiding the robot in understanding how its actions can affect the surroundings. This helps bridge the gap between high-level semantic knowledge and concrete actions, enabling more effective task planning.

Moving down to low-level motor control, incorporating physical inductive biases becomes crucial. These biases help the robot exploit its knowledge of the physical world, such as gravity, conservation of energy, object properties, and spatial relationships, to efficiently learn to perform precise motor control tasks. By combining hierarchical policies with representations from foundation models for semantic understanding (top level), affordances for task planning (mid level), and physical inductive biases for motor control (bottom), we might be able to build agents capable of learning and executing multiple novel tasks in unstructured environments.

In addition, despite the limitations of the sim-to-real paradigm, these methods offer a cheap alternative to generating massive datasets for training generalist robotics models. Using simulation, we could efficiently train a large set of robotic skills using well-designed rewards and simulation environments. These policies can be used for collecting large amounts of expert datasets in simulation and the real world. These data can ideally include multiple hardware platforms with the hope of building a multi-embodiement model for low-level control.

In summary, despite all the recent progress in robot learning, substantial efforts are yet required to deploy these methods on a large scale in the real world. At the moment, it seems like the two main axes for making progress in this field are scaling data acquisition and finding general inductive biases. Both aspects can be very beneficial for tackling the complexity of the problem, and their combination can potentially be even more powerful.

Part V

APPENDIX

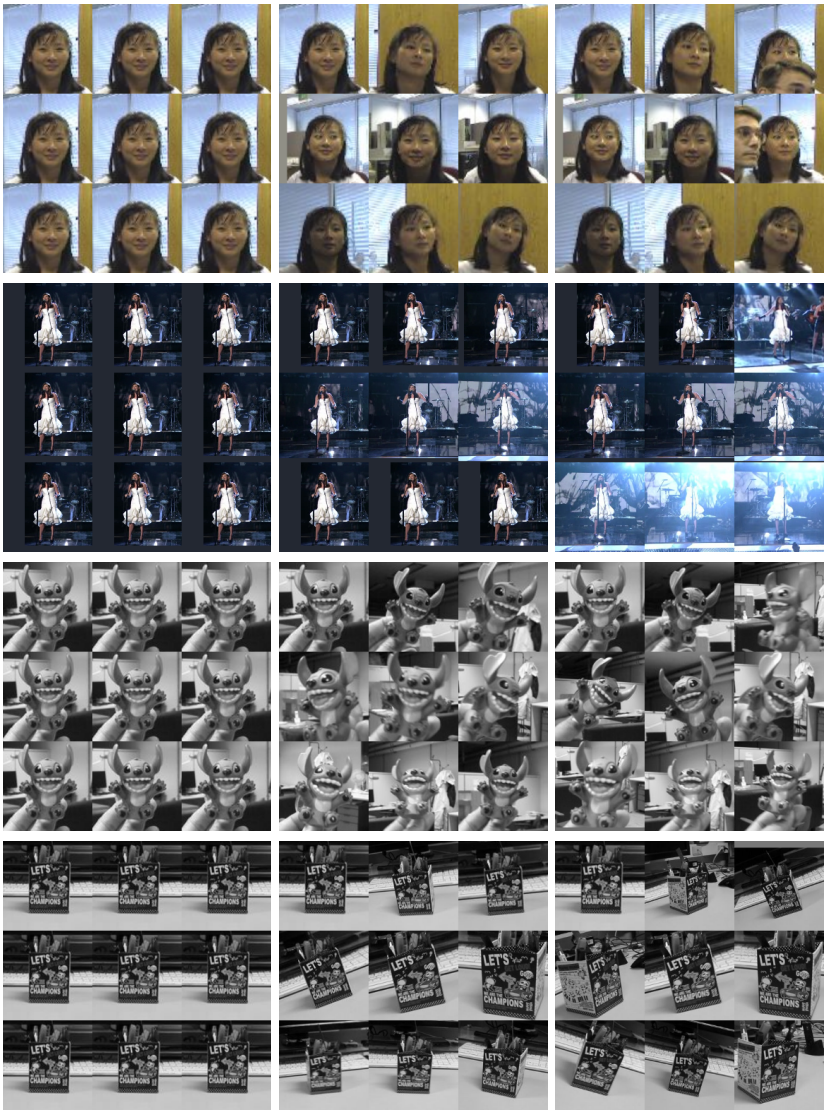
A

ROBOT STATE REPRESENTATIONS

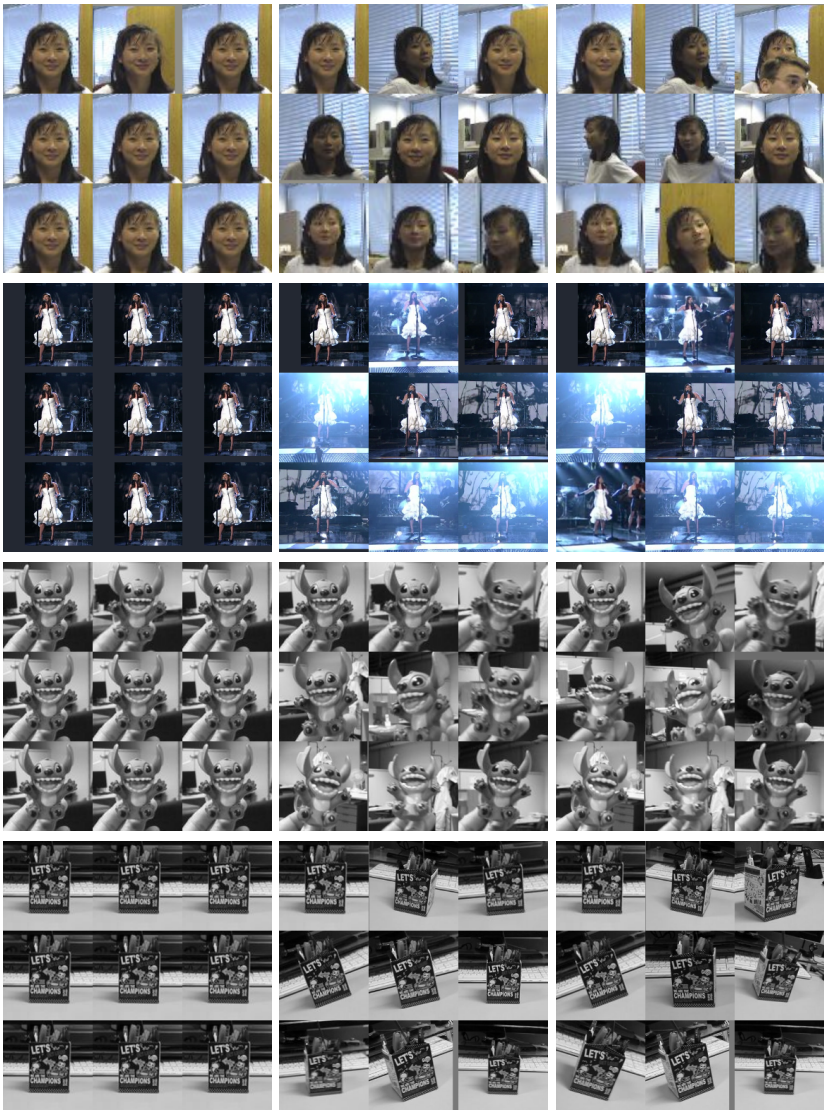
A.1 SUPERVISED STATE REPRESENTATION VIA ROBUST TRACKING

From left to right: template at the beginning, in the middle and at the end of the sequence. The sequences are from OTB2015 (Girl, Singer1, Toy, Vase) and we use SiamRPN for tracking. A.1.0.1 shows the templates using the dynamic lower bound, A.1.0.2 the ensemble lower bound.

A.1.0.1 *Dynamic Lower Bound*



A.1.0.2 *Ensemble-based Lower Bound*



A.2 EXPLORATION FOR STATE REPRESENTATION LEARNING

A.2.1 *Implementation Details & Hyperparameters*

For the encoder and decoder, we employ the architecture from [212]. Both consist of four convolutional layers with 3×3 kernels and 32 channels and use ReLU activations, except of the final deconvolution layer. Both networks use a stride of 1 for each layer except of the first of the encoder and the last of the decoder, which use stride 2.

For all shown experiments, we train with multiple seeds for each task. At the beginning of each seed, we pretrain the models with 1000 samples which we collect by rolling out random actions. Afterwards, we evaluate the model every 10 thousand environment steps over 10 episodes and report the average reward. The total number of episodes depends on the complexity of the task. All hyperparameters used in our experiments are summarized in Table A.1.

Table A.1: The hyperparameters used in our experiments.

Parameter	Setting
Batch size	128
Replay buffer capacity	80000
Discount γ	0.99
Hidden dimension	1024
Curious exploration probability p_c	0.2
Observation size	$84 \times 84 \times 3$
Frames stacked	3
Critic learning rate	10^{-3}
Critic target update frequency	2
Critic soft target update rate τ	0.01
Actor learning rate	10^{-3}
Actor update frequency	2
Actor log std bounds	[-10, 2]
Autoencoder learning rate	10^{-3}
Decoder update frequency	1
Temperature learning rate	10^{-4}
Init temperature	0.1

A.2.2 *Additional Results*

Here we show the exact values obtained for the SRL error visitation experiment in section 5.3.2. These values were roughly illustrated in figure 5.2 and exactly shown in Table A.2.

Table A.2: Mean, minimum and maximum SRL error encountered per step when using three different agents on six deepmind control suite tasks.

method/env	Vals	random	sac_ae	cure
reacher_easy	**Min**	0.0001	0.0001	0.0390
	Mean	0.0002	0.0004	0.0399
	Max	0.0003	0.0007	0.0424
ball_in_cup	**Min**	0.0005	0.0003	0.0762
	Mean	0.0006	0.0005	0.0774
	Max	0.0008	0.0007	0.0783
cartpole_swingup	**Min**	0.0002	0.0002	0.0531
	Mean	0.0002	0.0002	0.0540
	Max	0.0003	0.0003	0.0548
finger_spin	**Min**	0.0002	0.0004	0.0755
	Mean	0.0003	0.0012	0.0766
	Max	0.0004	0.0015	0.0774
finger_turn_easy	**Min**	0.0003	0.0003	0.0749
	Mean	0.0004	0.0005	0.0759
	Max	0.0006	0.0008	0.0769
reacher_hard	**Min**	0.0002	0.0002	0.0393
	Mean	0.0003	0.0004	0.0397
	Max	0.0008	0.0006	0.0404

B

ROBOT ACTION REPRESENTATIONS

B.1 THE ROLE OF THE ACTION SPACE

We provide more details concerning our environments. For the reaching task, we define the reward function

$$
\begin{aligned}
r_{\text{reach}}(s_t, a_t) &= \text{rew}_{\text{reach}}(s_t, a_t) - \text{pen}_{\text{reach}}(s_t, a_t) \\
\text{rew}_{\text{reach}} &= r_{\text{dist}} + r_{\text{exact}} \\
\text{pen}_{\text{reach}} &= r_{\text{vel}} + r_{\text{smooth}} + r_{\text{neutral}} + r_{\text{limit}} \\
r_{\text{dist}}(s_t, a_t) &= \lambda_r \cdot \frac{1}{1 + \|s_t - g\|_2^2} \\
r_{\text{exact}}(s_t, a_t) &= \mathbb{1}(\|s_t - g\|_2 < \epsilon)(\lambda_\epsilon + \frac{1}{1 + 100\dot{q}^2}) \\
r_{\text{vel}}(s_t, a_t) &= \lambda_q \cdot \|\dot{q}_t\|_2^2 \\
r_{\text{neutral}}(s_t, a_t) &= \lambda_n \cdot \|q_{\text{def}} - q\|_2 \\
r_{\text{limit}}(s_t, a_t) &= \lambda_l \cdot e^{-30(q - q_{\text{lim}})^2} \\
r_{\text{smooth}}(s_t, a_t) &= \lambda_s \cdot \|a_t - a_{t-1}\|_2,
\end{aligned}
$$

where we use the Euclidean norm as a distance metric. g is the end-effector goal position, q_{def} is the default joint positions vector, ϵ is a small positive constant, λ_r and λ_ϵ scale the reach and exact reach reward, λ_q and λ_s are positive scalars for the penalties on the velocity magnitude and smoothness of the action respectively and $\mathbb{1}$ is an indicator function, $lamda_n$ is a scalar for the penalties on divergence from the default joint position, and λ_l is a scalar for the joint limit avoidance penalty. For pushing we have a different reward,

$$
\begin{aligned}
r_{\text{push}}(s_t, a_t) &= \text{rew}_{\text{push}}(s_t, a_t) - \text{pen}_{\text{push}}(s_t, a_t) \\
\text{rew}_{\text{push}} &= r_{\text{dist}} + r_{\text{exact}} + r_{\text{push}} \\
\text{pen}_{\text{push}} &= r_{\text{vel}} + r_{\text{smooth}} + r_{\text{neutral}} + r_{\text{limit}} + r_{\text{col}} \\
r_{\text{col}}(s_t, a_t) &= \lambda_c \cdot \mathbb{1}(z_{ee} < 0.02),
\end{aligned}
$$

where λ_c is a scalar for the table collision penalty, z_{ee} is the end-effector's z-position, r_{dist} and r_{exact} use the object position as goal, and r_{push} is defined exactly as r_{dist}, but measures the distance between the object's position and the pushing goal position.

B.2 INDUCTIVE BIASES IN THE ACTION SPACE

B.2.1 *Joint Limit Avoidance*

Here we provide more information concerning the joint limit avoidance RMP policy. The metric matrix of the policy is defined as $\mathbf{G}_l := \texttt{diag}(b_1, b2, ...b7)$, where $b_i = (s(\alpha_u d + (1-\alpha_u)) + (1-s)(\alpha_l d + (1-\alpha_l)))^{-2}$ is the joint limit avoidance metric for the i-th joint q. In this formulation, $s = \frac{q-l_l}{l_u-l_l}$, $d = 4s(1-s)$, $\alpha_u = 1 - \exp(-\dot{q}_+^2/2\sigma^2)$ for $\sigma > 0$ and $\alpha_l = 1 - \exp(-\dot{q}_-^2/2\sigma^2)$. In this notation, $\dot{q}_+ = \dot{q}$ for $\dot{q} > 0$ and $\dot{q}_- = \dot{q}$ for $\dot{q} < 0$. The intuition behind this is the following: if the joint position is close to the limit l_u or l_l and the joint velocity heads to its limit, the metric goes to infinity; otherwise, the metric becomes 1. The definition of the damping matrix $\mathbf{B}_l$ is similar to the definitions in sections 7.2.1 and 7.2.2. Furthermore, since the curvature force near the joint limit is large enough, we define the potential field $\Phi_l := 0$. With the above-mentioned definitions, we can finally obtain $\mathbf{f}_l$ and $\mathbf{M}_l$ according to the GDS of $\mathcal{R}_l$.

B.2.2 *Implementation Details*

Previous work has shown that the performance of reinforcement learning algorithms is dependent on the implementation details [72, 155]. Thus, we provide further details about our method's models and training procedures for both representation (section B.2.2.1) and reinforcement learning (section B.2.2.2) for the sake of reproducibility.

B.2.2.1 *Representation Learning*

Our β-VAE encoder is a convolutional neural network (CNN) with five convolutional layers having in order 6, 32, 64, 128 and 256 channels. Each layer uses a rectified linear unit (ReLU) for activation and is followed by a batch normalization layer [83]. The mean and log standard deviation layers of the latent are linear and are also followed by a batch normalization layer. The mean layer uses a hyperbolic tangent (Tanh) activation.

For training, we use the Adam optimizer [97] with a learning rate of 0.001 and batches of size 128.

B.2.2.2 *Reinforcement Learning*

TD3 is an actor-critic method [55]. We use an actor with two hidden layers, each containing 100 neurons. The activation function of all layers is Tanh. The critic has two hidden layers, with 500 neurons each. For the critic, we use ReLU activations for hidden layers and no activation for the output. We use Adam to optimize both sets of parameters. The training hyperparameters are listed in the table below.

Table B.1: The hyperparameters used for RL.

Hyperparameter	Value
TD3 Policy noise	0.2
Max episode steps	400
Exploration noise	$0.5 \rightarrow 0.3$
Memory Size	300000
Batch size	64
Learning rate	$1e-3$

B.2.3 *Additional results*

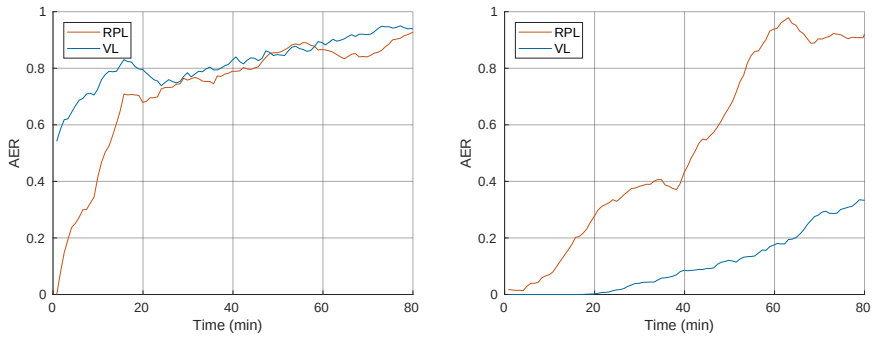
In this section we show some additional results related to experiments A and B (section B.2.3.1 and B.2.3.2) as well as to the latent model (section B.2.3.4).

B.2.3.1 *Experiment A*

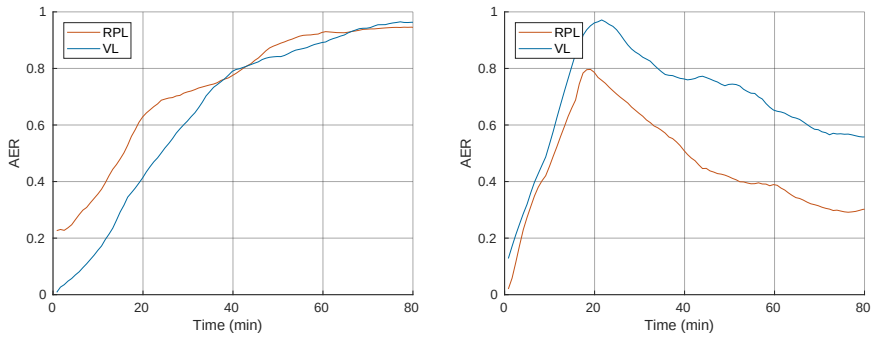
In addition to the previously shown total reward plot, here we show plots of the individual reward terms: $r_{collide}$, r_{goal}, r_{dist}, $r_{control}$. These values are especially interesting when comparing the policy learning with the baseline and without it. As before, we refer to those cases as **Residual Policy Learning (RPL)** and **Vanilla Learning (VL)** respectively. The results can be seen in figure B.1. As expected the goal reward for RPL is substantially higher than that of VL at all times. At the beginning of training, RPL leads to more collisions with the obstacle as the baseline policy guides it towards the goal. Subsequently, RPL starts with more negative reward $r_{collide}$ than VL. However, it manages after training to reach a similar level as VL. In contrast, the latter depending mostly on random actions barely hits the obstacle at these stages as it's not even directed towards the goal. This can be seen in the r_{goal} and r_{dist} plots. As for the control reward r_{ctrl}, it behaves similarly for both methods. However, it gets higher for vanilla learning after a while. This could be explained by the following: RPL based exploration leads at all times to higher goal reward than VL. The latter, barely reaching the goal, prefers to take smaller actions to increase r_{ctrl}.

B.2.3.2 *Experiment B*

In addition to the provided video here we show image sequences for our multiple obstacle avoidance results. We show a successful trial in figure B.2 and an unsuccessful trial in figure B.3.



(a) $r_{collide}$ (b) r_{goal}



(c) r_{dist} (d) r_{ctrl}

Figure B.1: Individual rewards for the policy training with a baseline (RPL) and without (VL).

Figure B.2: Multiple Obstacle Avoidance: Successful trial. The red circle is the goal.

Figure B.3: Multiple Obstacle Avoidance: Unsuccessful Trial. The red circle is the goal. The robot collides with the cylinder at the end of the execution.

Figure B.4: Single obstacle avoidance: cube. The red circle is the goal.

B.2.3.3 *Experiment C*

Although experiment B supports our claim of generalization, we conduct a further experiment to double-check our method's generalization ability. This experiment tests if the trained policy can generalize to unseen obstacles. We train our policy in single obstacle avoidance scenarios, with two types of obstacles: cuboid and sphere. After the training, we test the trained policy in scenarios with a cylinder obstacle. We evaluate the policy for 50 trials in the scenarios containing the unseen obstacle. We report that the success rate is 72%, which is similar to the success rate when evaluating the policy using the previously-seen obstacles. Besides the training curve, we also provide an image sequence of the execution. Figure B.4 and B.5 are evaluations of the policy on trained obstacles (cube and sphere). Figure B.6 shows a policy execution in the scenario with previously unseen obstacle (cylinder). The result of this experiment provides another evidence of the generalization ability of our method.

Figure B.5: Single obstacle avoidance: sphere. The red circle is the goal.

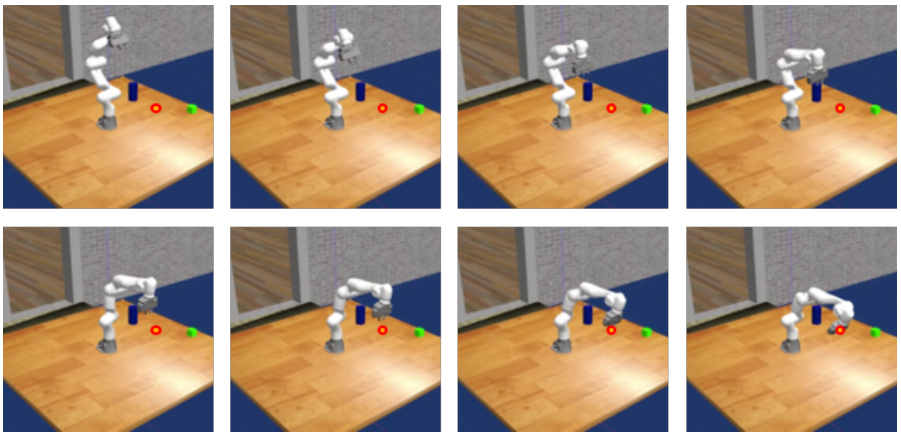


Figure B.6: Single obstacle avoidance: cylinder. The red circle is the goal.

Figure B.7: Images Sampled from latent z_0 and z_1. First two rows: $z_1 < 0$ and $z_0 \in [-1, 1]$; Last two rows, $z_1 > 0$ and $z_0 \in [-1, 1]$. z_0 and z_1 control together the y-position of the sphere.

B.2.3.4 *Visualizing the latent space*

Here we illustrate images sampled from our VAE's latent as to show the importance of such variables to the obstacle avoidance task. The samples are shown in figure B.7. Note that the sampled images are only blurry because of the down-sampling of the inputs images to the VAE.

B.3 CENTRAL LATENT ACTION SPACES

B.3.1 *Further Details*

B.3.1.1 *Models*

We provide further figures illustrating the computational architecture and graphical models related to the different components of the algorithm. Figure B.8 shows the grahical models of the policies involved in our method under full and partial observability.

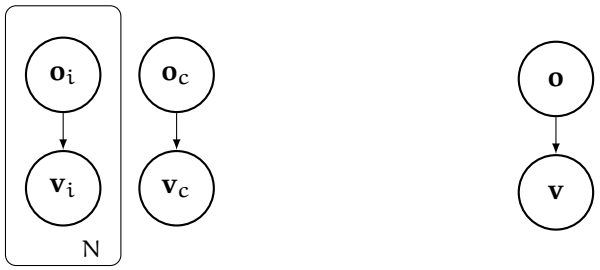


Figure B.8: Graphical models of the policies used by CLAS for the cases of partial (a) and full observability (b).

B.3.1.2 *Derivations*

Here we go over the derivation of equation (8.1) and provide more steps and explanations on how the derivation is performed:

$$
\begin{aligned}
p(\mathbf{u} \mid \mathbf{o}) &= \int p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v})\, p_\psi(\mathbf{v} \mid \mathbf{o})\, d\mathbf{v} \\
\ln p(\mathbf{u} \mid \mathbf{o}) &= \ln \int p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v})\, p_\psi(\mathbf{v} \mid \mathbf{o}) d\mathbf{v} \\
\ln p(\mathbf{u} \mid \mathbf{o}) &= \ln \int p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v})\, p_\psi(\mathbf{v} \mid \mathbf{o})\, \frac{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u})}{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u})}\, d\mathbf{v} \\
&\geqslant \int q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u}) \ln\Big(p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v}) \frac{p_\psi(\mathbf{v} \mid \mathbf{o})}{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u})} \Big)\, d\mathbf{v} \\
&= \mathbb{E}_{q_\phi(\mathbf{v}|\mathbf{o},\mathbf{u})} \left[\ln\Big(p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v}) \frac{p_\psi(\mathbf{v} \mid \mathbf{o})}{q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u})} \Big) \right] \\
&= \mathbb{E}_{q_\phi(\mathbf{v}|\mathbf{o},\mathbf{u})} \left[\ln p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v}) - \ln q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u}) + \ln p_\psi(\mathbf{v} \mid \mathbf{o}) \right] \\
&= \mathbb{E}_{q_\phi(\mathbf{v}|\mathbf{o},\mathbf{u})} [\ln p_\theta(\mathbf{u} \mid \mathbf{o}, \mathbf{v})] - \mathrm{KL}\big(q_\phi(\mathbf{v} \mid \mathbf{o}, \mathbf{u}) \,\big|\big|\, p_\psi(\mathbf{v} \mid \mathbf{o}) \big) \\
&= \mathcal{L}(\mathbf{u}, \theta, \phi, \psi \mid \mathbf{o}).
\end{aligned}
$$

The inequality step is based on Jensen's inequality, the pre-last step is due to the product and quotient rules of logarithms, and the last step is based on the definition of the KL divergence. The derivation is in line with the original lower bound derivation for variational autoencoders [98].

B.3.2 *Experiments*

B.3.2.1 *Setup*

Here we provide further details concerning our setup and experimental design, as to enable easy reproduction of our work.

The environments we used are based on joint velocity control action spaces. Each agent receives the corresponding robot's proprioceptive measurements, and the shared observation corresponds to object observations. For evaluation, we run each episode for 500 steps lead-

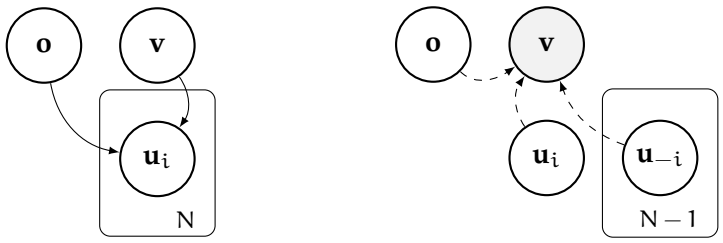


Figure B.9: Graphical models under full access to observations for all agents. (a) action generation, (b) latent action inference. During generation of actions $\mathbf{u}_i$ each agent i requires input from global observations $\mathbf{o}$ and central latent actions $\mathbf{v}$. In order to infer latent actions $\mathbf{v}$ information from all agents and the global observation is needed.

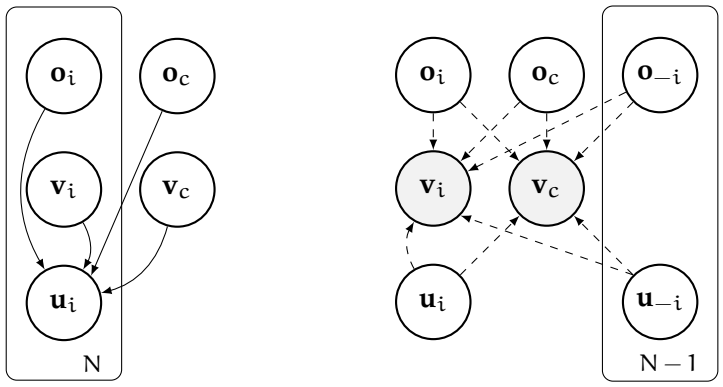
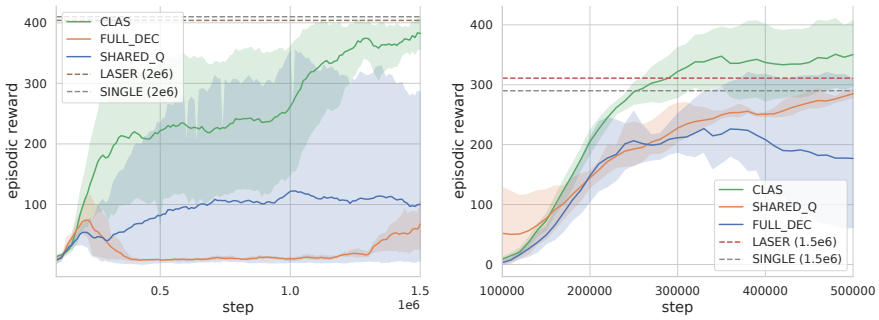


Figure B.10: Graphical models under partial agent observability. (a) action generation, (b) latent action inference. During generation of action $\mathbf{u}_i$ the input observation excludes all the other agents observations $\mathbf{o}_{-i}$ and latent actions $\mathbf{v}_{-i}$. Inference is done based on observations and actions from all agents.



dual-arm-lift four-arm-lift

Figure B.11: Results under full agent observability.

ing to maximal reward of 500. We run all evaluation experiments 10 times with different random seeds.

The reward used in the Lift environment is the following:

$$
\begin{aligned}
r_{\text{lift}} &= \max(d - 0.05, 0) \\
r_{\text{dir}} &= \begin{cases} 1, & \text{for } \cos(\alpha) \geqslant \cos(10°) \\ 0, & \text{for } \cos(\alpha) < \cos(10°) \end{cases} \\
r &= \frac{1}{3} \begin{cases} 3\, r_{\text{dir}}, & \text{for } d > 0.35 \\ 10\, r_{\text{dir}} + r_{\text{lift}}, & \text{for } d \leqslant 0.35, \end{cases}
\end{aligned} \tag{B.1}
$$

where d represents the distance between the surface and the pot, α the tilt angle of the pot.

B.3.2.2 *Results under full observability*

Here we study the performance of our method in the case where all agents have access to the full observation. We again compare to the same baselines. Similar to the results in section 8.4, our methods outperforms all MARL baselines in terms of final reward and sample efficiency. It also approaches the performance of the centralized single agents, and even outperforms them in four-arm-lift.

B.3.2.3 *Ablations*

In section 8.4, we showed that the shared latent actions are active during control. To make sure that the shared latent actions are not ignored during execution we perform the following experiment. We replace the shared latent actions with zeros during inference, and compare the achieved episodic reward to the standard case using our method. The results are in figure B.14. For the peg-in-hole environment, the difference in performance is minor. This is mainly due to the fact that this task does not necessarily involve objects that are independent of the robots. Instead the peg and hole are attached to

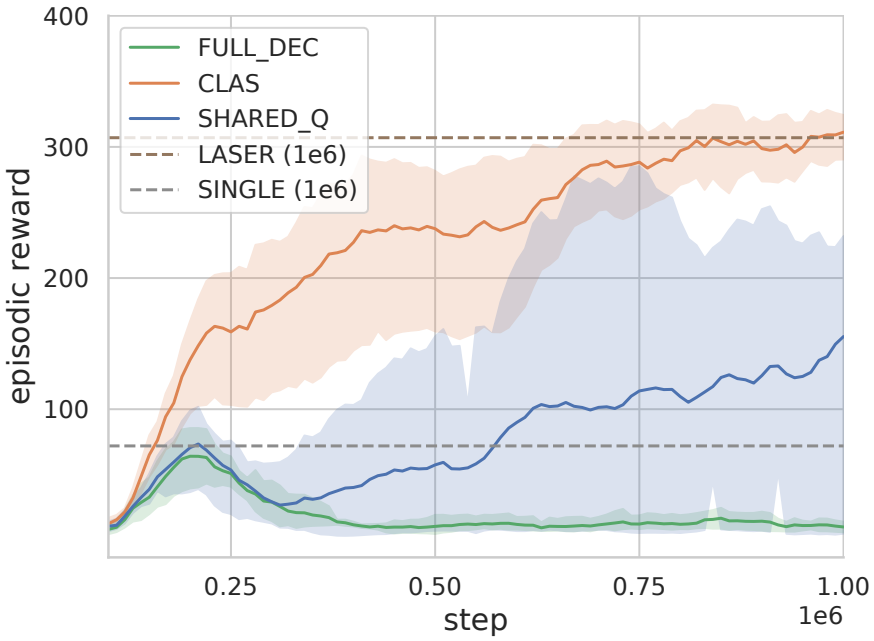


Figure B.12: Reward plots in the dual-arm-lift environments when using two different robots with different action spaces.

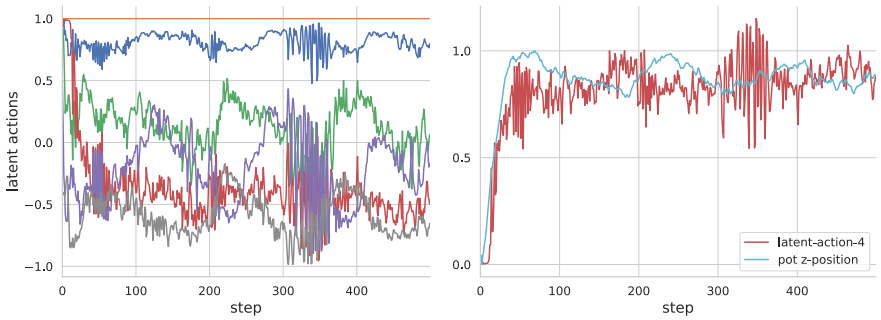
the corresponding robot. The improvement in results shown by our method in figure 8.5 is mainly due to the centralized training of the latent action space model. However, for the lifting environments, where a robots-independent object is to be manipulated, masking the shared latent action makes a huge difference. Namely, masking the shared latent actions with zeros leads to very low rewards. These two results indicate that our action space model maps robot actions into actions acting on the objects in some space.

B.3.2.4 *Results under Asymmetry of action spaces*

To check whether our approach is capable of handling asymmetric action spaces and multiple robots, we compare its performance to the baselines again in the dual-arm-lift environments. However, this time we use two different robots in the environment, namely we use a Panda and a Sawyer robot. The panda is equipped with a joint velocity action space and Sawyer with an operational space controller. The results are in figure B.12. `CLAS` is the only decentralized method that finds policies capable of lifting the pot, while the other two decentralized baselines as well as `SINGLE` struggle to do so.

B.3.2.5 *Coordination*

To demonstrate the coordination achieved by both agents we plot the desired joint velocity generated by the policy and the achieved joint velocity for both agents. This can be seen in figure B.15 and B.16. We



(a) (b)

Figure B.13: Central latent action trajectories for the lifting task. (a) Trajectories of all latent action dimensions. (b) Correlation between one shared latent action and the z position of the pot. The z-position trajectory of the pot (blue curve) follows the latent action trajectory (red curve).

notice that the dominant pattern across all plots is the diagonal. This shows that the policy outputs are used by both robots as opposed to having one robot being controlled by a policy, while the other being purely reactive and ignoring its policy outputs. The fourth joint is the only exception, where some policy outputs are ignored (mapped to zeros). However, also in this case, the most values fall on the diagonal.

B.3.2.6 *Qualitative Results*

Analyzing the central latent action space. To further validate our method, we examine the shared latent actions produced during evaluation. Figure B.13 shows trajectories of the shared latent actions produced by our model for the dual-arm-lift task. We observe that most shared latent action dimensions are active during control. One of the latent actions is constant during execution which illustrates that our approach could successfully recover a lower-dimension action space even when configured differently. Furthermore, we notice that the sequence of actions from the most varying latent action (in red) highly correlates with the z-position trajectory of the pot (figure B.13). The z-position follows the mentioned latent action with a slight time delay. In this case, this latent action represents desired z-positions of the pot needed to lift it. This is an interesting finding since our approach does not explicitly enforce any physical form or structure on the latent action space. The emergence of this property is purely due to the compression capabilities of variational autoencoders. Note that the plots in figure B.13 are qualitative results only meant to illustrate emergent latent actions spaces, and do not mean that our approach is interpretable.

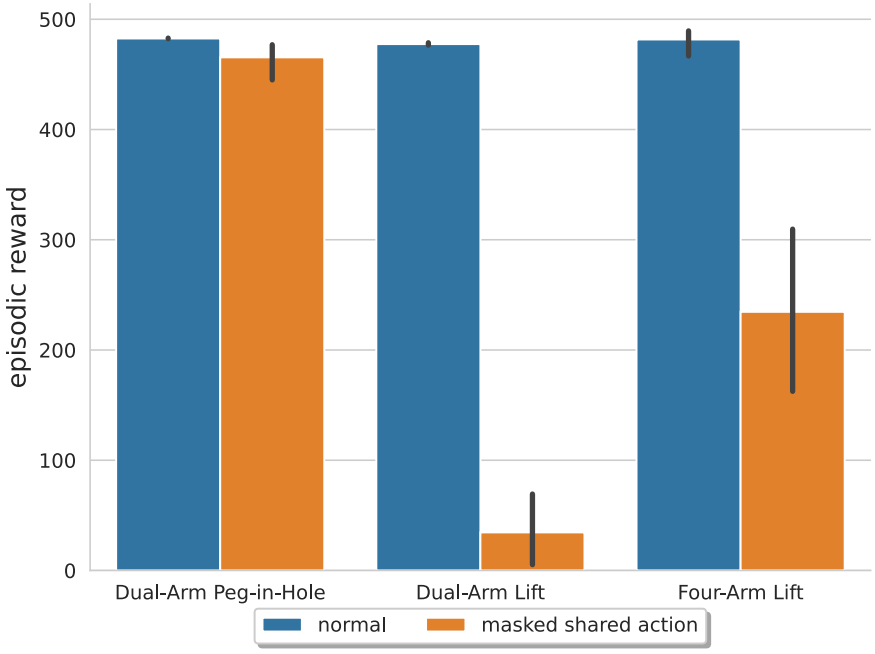


Figure B.14: Effect of masking the shared latent action on the achieved total reward.

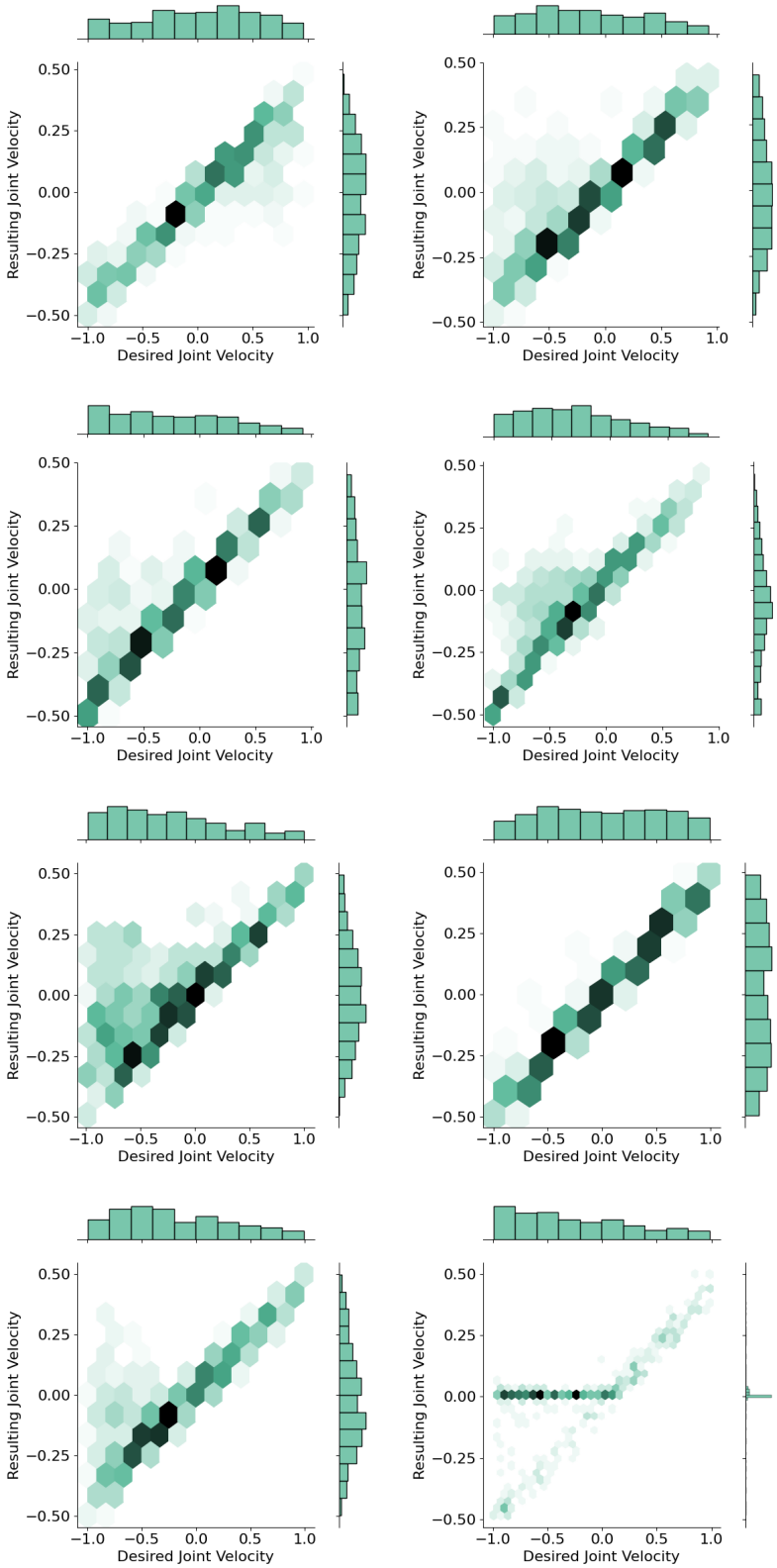


Figure B.15: Plots of the achieved joint velocity based on the commanded joint velocity for the two agents involved in the Lifting task. Each row indicates a joint [1-4].

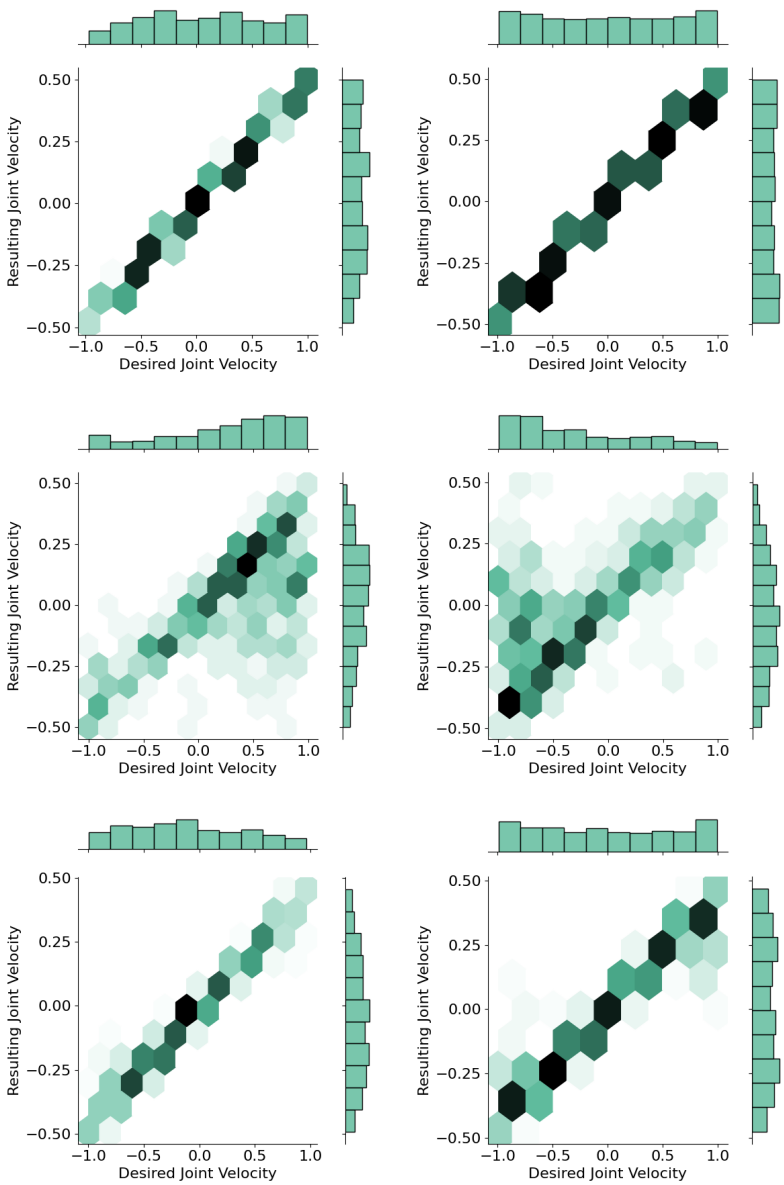


Figure B.16: Plots of the achieved joint velocity based on the commanded joint velocity for the two agents involved in the Lifting task. Each row indicates a joint [5-7].

BIBLIOGRAPHY

[1] Pieter Abbeel and Andrew Y Ng. "Apprenticeship learning via inverse reinforcement learning." In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[2] Joshua Achiam. "Spinning up in deep reinforcement learning." In: (2018).

[3] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. "Solving rubik's cube with a robot hand." In: *arXiv preprint arXiv:1910.07113* (2019).

[4] Elie Aljalbout, Ji Chen, Konstantin Ritt, Maximilian Ulmer, and Sami Haddadin. "Learning vision-based reactive policies for obstacle avoidance." In: *Proceedings of the 2020 Conference on Robot Learning*. Vol. 155. Proceedings of Machine Learning Research. PMLR, 2021.

[5] Elie Aljalbout, Felix Frank, Maximilian Karl, and Patrick van der Smagt. "On the role of the action space in robot manipulation learning and sim-to-real transfer." In: *IEEE Robotics and Automation Letters* (2024).

[6] Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, and Daniel Cremers. "Clustering with deep learning: Taxonomy and new methods." In: *arXiv preprint arXiv:1801.07648* (2018).

[7] Elie Aljalbout, Maximilian Karl, and Patrick van der Smagt. "CLAS: Coordinating Multi-Robot Manipulation with Central Latent Action Spaces." In: *Proceedings of The 5th Annual Learning for Dynamics and Control Conference (L4DC)*. 2023.

[8] Elie Aljalbout, Maximilian Ulmer, and Rudolph Triebel. "Making curiosity explicit in vision-based rl." In: *arXiv preprint arXiv:2109.13588* (2021).

[9] Elie Aljalbout, Maximilian Ulmer, and Rudolph Triebel. "Seeking visual discomfort: Curiosity-driven representations for reinforcement learning." In: *2022 International Conference on Robotics and Automation*. IEEE. 2022, pp. 3591–3597.

[10] Marvin Alles and Elie Aljalbout. "Learning to centralize dual-arm assembly." In: *Frontiers in Robotics and AI* 9 (2022), p. 830007.

[11] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. "Laser: Learning a latent action space for efficient reinforcement learning." In: *2021 IEEE International Conference on Robotics and Automation*. IEEE. 2021, pp. 6650–6656.

[12] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein. "Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs." In: *Autonomous Agents and Multi-Agent Systems* 21.3 (2010), pp. 293–320.

[13] Marcin Andrychowicz et al. "Learning dexterous in-hand manipulation." In: *Int. J. Robotics Res.* 39.1 (2020). DOI: 10.1177/0278364919887447. URL: https://doi.org/10.1177/0278364919887447.

[14] J.-A. M Assael, N. Wahlström, T. B. Schön, and M. P. Deisenroth. "Data-efficient learning of feedback policies from image pixels using deep dynamical models." In: *NIPS Deep Reinforcement Learning Workshop* (2015).

[15] Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, and Deepak Pathak. "Neural dynamic policies for end-to-end sensorimotor learning." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5058–5069.

[16] Dana H Ballard. "Modular learning in neural networks." In: *AAAI*. 1987, pp. 279–284.

[17] Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva TB, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. "Distributional policy gradients." In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SyZipzbCb.

[18] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. "Unifying count-based exploration and intrinsic motivation." In: *Advances in neural information processing systems* 29 (2016).

[19] Cristian Camilo Beltran-Hernandez, Damien Petit, Ixchel Georgina Ramirez-Alpizar, Takayuki Nishi, Shinichi Kikuchi, Takamitsu Matsubara, and Kensuke Harada. "Learning force control for contact-rich manipulation tasks with rigid position-controlled robots." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5709–5716.

[20] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives." In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult." In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[22] Daniel S Bernstein, Christopher Amato, Eric A Hansen, and Shlomo Zilberstein. "Policy iteration for decentralized control of Markov decision processes." In: *Journal of Artificial Intelligence Research* 34 (2009), pp. 89–132.

[23] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. "The complexity of decentralized control of Markov decision processes." In: *Mathematics of operations research* 27.4 (2002), pp. 819–840.

[24] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. "Fully-convolutional siamese networks for object tracking." In: *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part II 14*. Springer. 2016, pp. 850–865.

[25] Dimitri P Bertsekas. "Dynamic programming and optimal control 4th edition, volume ii." In: *Athena Scientific* (2015).

[26] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. "Model-predictive control via cross-entropy and gradient-based optimization." In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 277–286.

[27] Michael J Black and Allan D Jepson. "Eigentracking: Robust matching and tracking of articulated objects using a view-based representation." In: *International Journal of Computer Vision* 26 (1998), pp. 63–84.

[28] Miroslav Bogdanovic, Majid Khadiv, and Ludovic Righetti. "Learning variable impedance control for contact sensitive tasks." In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6129–6136. ISSN: 2377-3766. DOI: 10.1109/LRA.2020.3011379. (Visited on 05/10/2022).

[29] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. "Visual object tracking using adaptive correlation filters." In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2544–2550.

[30] Konstantinos Bousmalis, Giulia Vezzani, Dushyant Rao, Coline Devin, Alex X Lee, Maria Bauza, Todor Davchev, Yuxiang Zhou, Agrim Gupta, Akhil Raju, et al. "RoboCat: A self-improving foundation agent for robotic manipulation." In: *arXiv preprint arXiv:2306.11706* (2023).

[31] Michael Bowling and Manuela Veloso. "Multiagent learning using a variable learning rate." In: *Artificial Intelligence* 136.2 (2002), pp. 215–250.

[32] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. "Generating sentences from a continuous space." In: (2016), pp. 10–21.

[33] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. "Rt-1: Robotics transformer for real-world control at scale." In: *arXiv preprint arXiv:2212.06817* (2022).

[34] Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, et al. "Do as i can, not as i say: Grounding language in robotic affordances." In: *Conference on Robot Learning*. PMLR. 2023, pp. 287–318.

[35] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuvska. "Integrating state representation learning into deep reinforcement learning." In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1394–1401.

[36] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. "Exploration by random network distillation." In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=H1lJJnR5Ym.

[37] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. "Multi-agent reinforcement learning: A review of challenges and applications." In: *Applied Sciences* 11.11 (2021), p. 4948.

[38] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. "Closing the sim-to-real loop: Adapting simulation randomization with real world experience." In: *2019 International Conference on Robotics and Automation*. IEEE. 2019, pp. 8973–8979.

[39] Boyu Chen, Peixia Li, Lei Bai, Lei Qiao, Qiuhong Shen, Bo Li, Weihao Gan, Wei Wu, and Wanli Ouyang. "Backbone is all your need: A simplified architecture for visual object tracking." In: *European Conference on Computer Vision*. Springer. 2022, pp. 375–392.

[40] Xin Chen, Houwen Peng, Dong Wang, Huchuan Lu, and Han Hu. "SeqTrack: Sequence to sequence learning for visual object tracking." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14572–14581.

[41] Ching-An Cheng, Mustafa Mukadam, Jan Issac, Stan Birchfield, Dieter Fox, Byron Boots, and Nathan Ratliff. "RMPflow: A computational graph for automatic motion policy generation." In: *International Workshop on the Algorithmic Foundations of Robotics*. Springer. 2018.

[42] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification." In: *2005 IEEE computer society conference on computer vision and pattern recognition*. Vol. 1. IEEE. 2005, pp. 539–546.

[43] Yutao Cui, Cheng Jiang, Limin Wang, and Gangshan Wu. "Mixformer: End-to-end tracking with iterative mixed attention." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 13608–13618.

[44] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. "Tarmac: Targeted multi-agent communication." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1538–1546.

[45] Marc Deisenroth and Carl E Rasmussen. "PILCO: A model-based and data-efficient approach to policy search." In: *Proceedings of the 28th International Conference on machine learning*. 2011, pp. 465–472.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of deep bidirectional transformers for language understanding." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/V1/N19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

[47] Jilles Steeve Dibangoye, Christopher Amato, Olivier Buffet, and François Charpillet. "Optimally solving Dec-POMDPs as continuous-state MDPs." In: *Journal of Artificial Intelligence Research* 55 (2016), pp. 443–497.

[48] Jilles Dibangoye and Olivier Buffet. "Learning to act in decentralized partially observable MDPs." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1233–1242.

[49] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. "An image is worth 16x16 words: transformers for image recognition at scale." In: *International Conference on Learning Representations*. 2021.

[50] Danny Driess et al. "PaLM-E: An embodied multimodal language model." In: *Proceedings of the 40th International Conference on Machine Learning*. Honolulu, Hawaii, USA, 2023.

[51] Helei Duan, Jeremy Dao, Kevin Green, Taylor Apgar, Alan Fern, and Jonathan Hurst. "Learning task space actions for bipedal locomotion." In: *2021 IEEE International Conference on Robotics and Automation*. IEEE. 2021, pp. 1276–1282.

[52] Qi Feng, Vitaly Ablavsky, Qinxun Bai, Guorong Li, and Stan Sclaroff. "Real-time visual object tracking with natural language description." In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 700–709.

[53] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. "Counterfactual multi-agent policy gradients." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[54] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[55] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing function approximation error in actor-critic methods." In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018.

[56] Aditya Ganapathi, Pete Florence, Jake Varley, Kaylee Burns, Ken Goldberg, and Andy Zeng. "Implicit kinematic policies: Unifying joint and cartesian action spaces in end-to-end robot learning." In: *2022 International Conference on Robotics and Automation*. IEEE. 2022, pp. 2656–2662.

[57] Shenyuan Gao, Chunluan Zhou, Chao Ma, Xinggang Wang, and Junsong Yuan. "Aiatrack: Attention in attention for transformer visual tracking." In: *European Conference on Computer Vision*. Springer. 2022, pp. 146–164.

[58] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets." In: *Advances in neural information processing systems* 27 (2014).

[59] Ross Goroshin, Michael F Mathieu, and Yann LeCun. "Learning to linearize under uncertainty." In: *Advances in neural information processing systems* 28 (2015).

[60] Manuel Graña, Borja Fernandez-Gauna, and Jose Manuel Lopez-Guede. "Cooperative multi-agent reinforcement learning for multi-component robotic systems: guidelines for future research." In: *Paladyn* 2.2 (2011), pp. 71–81.

[61] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. "Coordinated reinforcement learning." In: *International conference on machine learning*. Vol. 2. Citeseer. 2002, pp. 227–234.

[62] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning." In: *International conference on autonomous agents and multiagent systems*. Springer. 2017, pp. 66–83.

[63] Mehmet Serdar Guzel and Robert Bicker. "Vision based obstacle avoidance techniques." In: *Recent advances in mobile robotics* (2011).

[64] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic Actor." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 1861–1870. URL: https://proceedings.mlr.press/v80/haarnoja18b.html.

[65] Sami Haddadin, Holger Urbanek, Sven Parusel, Darius Burschka, Jürgen Roßmann, Alin Albu-Schäffer, and Gerd Hirzinger. "Real-time reactive motion generation based on variable attractor dynamics and shaped velocities." In: *International Conference on Intelligent Robots and Systems*. 2010.

[66] Philip Haeusser, Johannes Plapp, Vladimir Golkov, Elie Aljalbout, and Daniel Cremers. "Associative deep clustering: Training a classification network with no labels." In: *Pattern Recognition: 40th German Conference, GCPR 2018, Stuttgart, Germany, October 9-12, 2018, Proceedings 40*. 2019.

[67] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. "Dream to control: Learning behaviors by latent imagination." In: *International Conference on Learning Representations*. 2019.

[68] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning latent dynamics for planning from pixels." In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2555–2565.

[69] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. "Dextreme: Transfer of agile in-hand manipulation from simulation to reality." In: *2023 IEEE International Conference on Robotics and Automation*. IEEE. 2023, pp. 5977–5984.

[70] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. "Masked autoencoders are scalable vision learners." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 16000–16009.

[71] David Held, Sebastian Thrun, and Silvio Savarese. "Learning to track at 100 fps with deep regression networks." In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, pp. 749–765.

[72] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. "Deep reinforcement learning that matters." In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[73] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework." In: *International conference on learning representations*. 2016.

[74] Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. "Darla: Improving zero-shot transfer in reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1480–1490.

[75] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[76] Elad Hoffer and Nir Ailon. "Deep metric learning using triplet network." In: *Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3*. Springer. 2015, pp. 84–92.

[77] Andreas Hofhauser, Carsten Steger, and Nassir Navab. "Edge-based template matching and tracking for perspectively distorted planar objects." In: *International Symposium on Visual Computing*. Springer. 2008, pp. 35–44.

[78] Zhibin Hong, Zhe Chen, Chaohui Wang, Xue Mei, Danil Prokhorov, and Dacheng Tao. "Multi-store tracker (muster): A cognitive psychology inspired approach to object tracking." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 749–758.

[79] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. "Vime: Variational information maximizing exploration." In: *Advances in neural information processing systems* 29 (2016).

[80] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents." In: *International Conference on Machine Learning*. PMLR. 2022, pp. 9118–9147.

[81] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. "Learning agile and dynamic motor skills for legged robots." In: *Science Robotics* 4.26 (2019), eaau5872.

[82] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. "Dynamical movement primitives: learning attractor models for motor behaviors." In: *Neural computation* 25.2 (2013), pp. 328–373.

[83] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[84] Junior Costa de Jesus, Victor Augusto Kich, Alisson Henrique Kolling, Ricardo Bedin Grando, Marco Antonio de Souza Leite Cuadros, and Daniel Fernando Tello Gamarra. "Soft actor-critic for navigation of mobile robots." In: *Journal of Intelligent & Robotic Systems* 102.2 (2021), p. 31.

[85] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. "Residual reinforcement learning for robot control." In: *2019 International Conference on Robotics and Automation*. IEEE. 2019.

[86] Rico Jonschkowski and Oliver Brock. "Learning state representations with robotic priors." In: *Autonomous Robots* 39.3 (2015), pp. 407–428.

[87] Frédéric Jurie, Michel Dhome, et al. "Real time robust template matching." In: *BMVC*. Vol. 2002. 2002, pp. 123–132.

[88] Sham Kakade and John Langford. "Approximately optimal approximate reinforcement learning." In: *Proceedings of the Nineteenth International Conference on Machine Learning*. 2002, pp. 267–274.

[89] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. "Learning force control policies for compliant manipulation." In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 4639–4644.

[90] Siddharth Karamcheti, Megha Srivastava, Percy Liang, and Dorsa Sadigh. "LILA: Language-informed latent actions." In: *5th Annual Conference on Robot Learning*. 2021. URL: https://openreview.net/forum?id=_lkBGOctkip.

[91] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. "Deep variational bayes filters: Unsupervised learning of state space models from raw data." In: *International Conference on Learning Representations*. 2016.

[92] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. "A benchmark comparison of learned control policies for agile quadrotor flight." In: *2022 International Conference on Robotics and Automation*. IEEE. 2022, pp. 10504–10510.

[93] Seyed Mohammad Khansari-Zadeh and Aude Billard. "A dynamical system approach to real-time obstacle avoidance." In: *Autonomous Robots* (2012).

[94] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots." In: *Autonomous robot vehicles*. Springer, 1986.

[95] Oussama Khatib. "A unified approach for motion and force control of robot manipulators: The operational space formulation." In: *IEEE Journal on Robotics and Automation* (1987).

[96] Beobkyoon Kim, Cheongjae Jang, and Jisoo Hong. "Geometric algorithms for robot dynamics: A tutorial review." In: ().

[97] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[98] Diederik P Kingma and Max Welling. "Auto-encoding variational bayes." In: *International Conference on Learning Representations*. 2014.

[99] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. "Siamese neural networks for one-shot image recognition." In: *ICML deep learning workshop*. Vol. 2. 1. Lille. 2015.

[100] Nathan Koenig and Andrew Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator." In: *International Conference on Intelligent Robots and Systems*. IEEE. 2004.

[101] Matej Kristan, Jiri Matas, Alevs Leonardis, Tomávs Vojívr, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka vCehovin. "A novel performance evaluation methodology for single-target trackers." In: *IEEE transactions on pattern analysis and machine intelligence* 38.11 (2016), pp. 2137–2155.

[102] Matej Kristan, Jiri Matas, Alevs Leonardis, Tomávs Vojívr, Roman Pflugfelder, Gustavo Fernandez, Georg Nebehay, Fatih Porikli, and Luka vCehovin. "A novel performance evaluation methodology for single-target trackers." In: *IEEE transactions on pattern analysis and machine intelligence* 38.11 (2016), pp. 2137–2155.

[103] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012).

[104] Sascha Lange and Martin Riedmiller. "Deep auto-encoder neural networks in reinforcement learning." In: *The 2010 International Joint Conference on Neural Networks*. IEEE. 2010, pp. 1–8.

[105] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. "Curl: Contrastive unsupervised representations for reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5639–5650.

[106] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. "Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 741–752.

[107] Hankyeol Lee, Seokeon Choi, and Changick Kim. "A memory model based on the siamese network for long-term tracking." In: *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*. 2018, pp. 0–0.

[108] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks." In: *2019 International Conference on Robotics and Automation*. IEEE. 2019, pp. 8943–8950.

[109] Youngwoon Lee, Jingyun Yang, and Joseph J. Lim. "Learning to coordinate manipulation skills via skill behavior diversification." In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=ryxB2lBtvH.

[110] Timothée Lesort, Natalia Díaz-Rodríguez, Jean-Francois Goudou, and David Filliat. "State representation learning for control: An overview." In: *Neural Networks* 108 (2018), pp. 379–392.

[111] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. "End-to-end training of deep visuomotor policies." In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 1334–1373.

[112] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, J Siamrpn+ Yan, et al. "Evolution of siamese visual tracking with very deep networks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA*. 2019, pp. 15–20.

[113] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing, and Junjie Yan. "Siamrpn++: Evolution of siamese visual tracking with very deep networks." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4282–4291.

[114] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu, and Xiaolin Hu. "High performance visual tracking with siamese region proposal network." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8971–8980.

[115] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. "Code as policies: Language model programs for embodied control." In: *2023 IEEE International Conference on Robotics and Automation*. IEEE. 2023, pp. 9493–9500.

[116] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning." In: *4th International Conference on Learning Representations*. 2016.

[117] Luyu Liu, Qianyuan Liu, Yong Song, Bao Pang, Xianfeng Yuan, and Qingyang Xu. "A collaborative control method of dual-arm robots based on deep reinforcement learning." In: *Applied Sciences* 11.4 (2021), p. 1816.

[118] Minghuan Liu, Ming Zhou, Weinan Zhang, Yuzheng Zhuang, Jun Wang, Wulong Liu, and Yong Yu. "Multi-agent interactions modeling with correlated policies." In: *International Conference on Learning Representations*. 2019.

[119] Xingyu Liu and Kris M. Kitani. "V-MAO: Generative modeling for multi-arm manipulation of articulated objects." In: *Proceedings of the 5th Conference on Robot Learning*. Ed. by Aleksandra Faust, David Hsu, and Gerhard Neumann. Vol. 164. Proceedings of Machine Learning Research. PMLR, 2022, pp. 287–296. URL: https://proceedings.mlr.press/v164/liu22a.html.

[120] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. "Multi-agent actor-critic for mixed cooperative-competitive environments." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 6382–6393. (Visited on 05/10/2022).

[121] Jianlan Luo, Eugen Solowjow, Chengtao Wen, Juan Aparicio Ojea, Alice M Agogino, Aviv Tamar, and Pieter Abbeel. "Reinforcement learning on variable impedance controller for high-precision robotic assembly." In: *2019 International Conference on Robotics and Automation*. IEEE. 2019, pp. 3080–3087.

[122] Yi Ma, Stefano Soatto, Jana Kovsecká, and Shankar Sastry. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. Springer, 2004.

[123] Yuntao Ma, Farbod Farshidian, Takahiro Miki, Joonho Lee, and Marco Hutter. "Combining learning-based locomotion policy with model-based manipulation for legged mobile manipulators." In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2377–2384.

[124] Viktor Makoviychuk et al. "Isaac gym: High performance GPU based physics simulation for robot learning." In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=fgFBtYgJQX_.

[125] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. "What Matters in Learning from Offline Human Demonstrations for Robot Manipulation." In: *Conference on Robot Learning*. PMLR. 2022, pp. 1678–1690.

[126] Roberto Martín-Martín, Michelle A Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. "Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2019, pp. 1010–1017.

[127] Jan Matas, Stephen James, and Andrew J Davison. "Sim-to-real reinforcement learning for deformable object manipulation." In: *Conference on Robot Learning*. PMLR. 2018, pp. 734–743.

[128] Shaunak A Mehta, Sagar Parekh, and Dylan P Losey. "Learning latent actions without human demonstrations." In: *2022 International Conference on Robotics and Automation*. IEEE. 2022, pp. 7437–7443.

[129] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed representations of words and phrases and their compositionality." In: *Advances in neural information processing systems* 26 (2013).

[130] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. "A graph placement methodology for fast chip design." In: *Nature* 594.7862 (2021), pp. 207–212.

[131] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (2013).

[132] Daniel Mohr. "Continuous Edge gradient-based template matching for articulated objects." In: (2009).

[133] Anahita Mohseni-Kabir, David Isele, and Kikuo Fujimura. "Interaction-aware multi-agent reinforcement learning for mobile agents with individual goals." In: *2019 International Conference on Robotics and Automation*. IEEE. 2019, pp. 3370–3376.

[134] Jelle Munk, Jens Kober, and Robert Babuvska. "Learning state representation for deep actor-critic control." In: *2016 IEEE 55th conference on decision and control*. IEEE. 2016, pp. 4667–4673.

[135] Andrew Y Ng and Stuart J Russell. "Algorithms for inverse reinforcement learning." In: *Proceedings of the Seventeenth International Conference on Machine Learning*. 2000, pp. 663–670.

[136] Hieu Tat Nguyen, Marcel Worring, and Rein Van Den Boomgaard. "Occlusion robust adaptive template tracking." In: *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*. Vol. 1. IEEE. 2001, pp. 678–683.

[137] Yaru Niu, Rohan R Paleja, and Matthew C Gombolay. "Multi-agent graph-attention communication and teaming." In: *International conference on autonomous agents and multiagent systems*. 2021, pp. 964–973.

[138] Clark F Olson and Daniel P Huttenlocher. "Automatic target recognition by matching oriented edge pixels." In: *IEEE Transactions on image processing* 6.1 (1997), pp. 103–113.

[139] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding." In: *arXiv preprint arXiv:1807.03748* (2018).

[140] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. "Deep exploration via bootstrapped DQN." In: *Advances in neural information processing systems* 29 (2016).

[141] Georg Ostrovski, Marc G Bellemare, Aäron Oord, and Rémi Munos. "Count-based exploration with neural density models." In: *International conference on machine learning*. PMLR. 2017, pp. 2721–2730.

[142] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. "Open x-embodiment: Robotic learning datasets and rt-x models." In: *arXiv preprint arXiv:2310.08864* (2023).

[143] Èric Pairet, Paola Ardón, Frank Broz, Michael Mistry, and Yvan Petillot. "Learning and generalisation of primitives skills towards robust dual-arm manipulation." In: *arXiv preprint arXiv:1904.01568* (2019).

[144] Christos H Papadimitriou and John N Tsitsiklis. "The complexity of Markov decision processes." In: *Mathematics of operations research* 12.3 (1987), pp. 441–450.

[145] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. "Probabilistic movement primitives." In: *Advances in neural information processing systems* 26 (2013).

[146] Adam Paszke et al. "PyTorch: An imperative style, high-performance deep learning library." In: *Advances in Neural Information Processing Systems*. 2019.

[147] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. "Curiosity-driven exploration by self-supervised prediction." In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2778–2787.

[148] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Sim-to-real transfer of robotic control with dynamics randomization." In: *2018 IEEE international conference on robotics and automation*. IEEE. 2018, pp. 3803–3810.

[149] Xue Bin Peng and Michiel Van De Panne. "Learning locomotion skills using deeprl: Does the choice of action space matter?" In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–13.

[150] Boris T Polyak and Anatoli B Juditsky. "Acceleration of stochastic approximation by averaging." In: *SIAM journal on control and optimization* 30.4 (1992), pp. 838–855.

[151] Arnu Pretorius, Scott Cameron, Andries Petrus Smit, Elan van Biljon, Lawrence Francis, Femi Azeez, Alexandre Laterre, and Karim Beguir. "Learning to communicate through imagination with model-based deep multi-agent reinforcement learning." In: (2020).

[152] Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, Julian Ibarz, and Sergey Levine. "Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods." In: *2018 IEEE International Conference on Robotics and Automation*. IEEE. 2018, pp. 6284–6291.

[153] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. "Improving language understanding by generative pre-training." In: (2018).

[154] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. "Modeling others using oneself in multi-agent reinforcement learning." In: *International conference on machine learning*. PMLR. 2018, pp. 4257–4266.

[155] Nirnai Rao, Elie Aljalbout, Axel Sauer, and Sami Haddadin. "How to make deep RL work in practice." In: *arXiv preprint arXiv:2010.13083* (2020).

[156] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4295–4304.

[157] Nathan D Ratliff, Jan Issac, Daniel Kappler, Stan Birchfield, and Dieter Fox. "Riemannian motion policies." In: *arXiv preprint arXiv:1801.02854* (2018).

[158] Nathan Ratliff, Marc Toussaint, and Stefan Schaal. "Understanding the geometry of workspace obstacles in motion optimization." In: *International Conference on Robotics and Automation*. IEEE. 2015.

[159] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems* 28 (2015).

[160] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic backpropagation and approximate inference in deep generative models." In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[161] Axel Sauer, Elie Aljalbout, and Sami Haddadin. "Tracking holistic object representations." In: *British Machine Vision Conference*. 2019.

[162] Stefan Schaal. "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics." In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[163] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. "Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (2015).

[164] Juergen Schmidhuber. "Curious model-building control systems." In: *Proc. international joint conference on neural networks*. 1991, pp. 1458–1463.

[165] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1889–1897. URL: https://proceedings.mlr.press/v37/schulman15.html.

[166] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." In: *arXiv preprint arXiv:1707.06347* (2017).

[167] Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. "State entropy maximization with random encoders for efficient exploration." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9443–9454.

[168] Pratyusha Sharma, Lekha Mohan, Lerrel Pinto, and Abhinav Gupta. "Multiple interactions made easy (mime): Large scale demonstrations data for imitation." In: *Conference on robot learning*. PMLR. 2018, pp. 906–915.

[169] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. "Loss is its own reward: Self-supervision for reinforcement learning." In: *arXiv preprint arXiv:1612.07307* (2016).

[170] Lucy Xiaoyang Shi, Archit Sharma, Tony Z Zhao, and Chelsea Finn. "Waypoint-based imitation learning for robotic manipulation." In: *arXiv preprint arXiv:2307.14326* (2023).

[171] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587 (2016), pp. 484–489.

[172] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. "Learning when to communicate at scale in multiagent cooperative and competitive tasks." In: *International Conference on Learning Representations*. 2018.

[173] Laura Smith, Ilya Kostrikov, and Sergey Levine. "A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning." In: *arXiv preprint arXiv:2208.07860* (2022).

[174] Carsten Steger. "Occlusion, clutter, and illumination invariant object recognition." In: *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* 34.3/A (2002), pp. 345–350.

[175] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. "Decoupling representation learning from reinforcement learning." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9870–9879.

[176] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and policy considerations for deep learning in NLP." In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2019.

[177] Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. "End-to-end memory networks." In: *Advances in neural information processing systems* 28 (2015).

[178] Sainbayar Sukhbaatar, arthur szlam, and Rob Fergus. "Learning multiagent communication with backpropagation." In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016.

[179] Peter Sunehag et al. "Value-decomposition networks for cooperative multi-agent learning based on team reward." In: *International conference on autonomous agents and multiagent systems*. 2018, pp. 2085–2087. URL: http://dl.acm.org/citation.cfm?id=3238080.

[180] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems* 27 (2014).

[181] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[182] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation." In: *Advances in neural information processing systems* 12 (1999).

[183] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. "Sim-to-real: Learning agile locomotion for quadruped robots." In: *Robotics: Science and Systems* (2018).

[184] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. "IndustReal: Transferring contact-rich assembly tasks from simulation to reality." In: *arXiv preprint arXiv:2305.17110* (2023).

[185] Ran Tao, Efstratios Gavves, and Arnold WM Smeulders. "Siamese instance search for tracking." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1420–1429.

[186] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. "Deepmind control suite." In: *arXiv preprint arXiv:1801.00690* (2018).

[187] Emanuel Todorov, Tom Erez, and Yuval 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.

[188] George E Uhlenbeck and Leonard S Ornstein. "On the theory of the Brownian motion." In: *Physical review* (1930).

[189] Maximilian Ulmer, Elie Aljalbout, Sascha Schwarz, and Sami Haddadin. "Learning robotic manipulation skills using an adaptive force-impedance action space." In: *arXiv preprint arXiv:2110.09904* (2021).

[190] Jack Valmadre, Luca Bertinetto, Joao F Henriques, Ran Tao, Andrea Vedaldi, Arnold WM Smeulders, Philip HS Torr, and Efstratios Gavves. "Long-term tracking in the wild: A benchmark." In: *Proceedings of the European conference on computer vision*. 2018, pp. 670–685.

[191] Jack Valmadre, Luca Bertinetto, Joao Henriques, Andrea Vedaldi, and Philip HS Torr. "End-to-end representation learning for correlation filter based tracking." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2805–2813.

[192] Herke Van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. "Stable reinforcement learning with autoencoders for tactile and visual data." In: *2016 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2016, pp. 3928–3934.

[193] Patrick Varin, Lev Grossman, and Scott Kuindersma. "A comparison of action spaces for learning manipulation tasks." In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2019, pp. 6015–6021.

[194] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in neural information processing systems* 30 (2017).

[195] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. "Chatgpt for robotics: Design principles and model abilities." In: *Microsoft Auton. Syst. Robot. Res* 2 (2023), p. 20.

[196] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. "From pixels to torques: Policy learning with deep dynamical models." In: *arXiv preprint arXiv:1502.02251* (2015).

[197] Qiang Wang, Li Zhang, Luca Bertinetto, Weiming Hu, and Philip HS Torr. "Fast online object tracking and segmentation: A unifying approach." In: *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2019, pp. 1328–1338.

[198] Xiaolong Wang and Abhinav Gupta. "Unsupervised learning of visual representations using videos." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2794–2802.

[199] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Advances in neural information processing systems* 28 (2015).

[200] Michael Watterson, Sikang Liu, Ke Sun, Trey Smith, and Vijay Kumar. "Trajectory optimization on manifolds with applications to SO (3) and R3XS2." In: *Robotics: Science and Systems*. 2018.

[201] Xing Wei, Yifan Bai, Yongchao Zheng, Dahu Shi, and Yihong Gong. "Autoregressive visual tracking." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 9697–9706.

[202] Jason Weston, Sumit Chopra, and Antoine Bordes. "Memory networks." In: *arXiv preprint arXiv:1410.3916* (2014).

[203] Daniël Willemsen, Mario Coppola, and Guido CHE de Croon. "MAMBPO: Sample-efficient multi-robot reinforcement learning using learned world models." In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2021, pp. 5635–5640.

[204] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. "Information theoretic MPC for model-based reinforcement learning." In: *2017 IEEE International Conference on Robotics and Automation*. IEEE. 2017, pp. 1714–1721.

[205] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* 8 (1992), pp. 229–256.

[206] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas Funkhouser. "Tidybot: Personalized robot assistance with large language models." In: *arXiv preprint arXiv:2305.05658* (2023).

[207] Qiangqiang Wu, Tianyu Yang, Ziquan Liu, Baoyuan Wu, Ying Shan, and Antoni B Chan. "DropMAE: Masked autoencoders with spatial-attention dropout for tracking tasks." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14561–14571.

[208] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. "Object Tracking Benchmark." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1834–1848. DOI: 10.1109/TPAMI.2014.2388226.

[209] Zhaoming Xie, Patrick Clary, Jeremy Dao, Pedro Morais, Jonanthan Hurst, and Michiel Panne. "Learning locomotion skills for cassie: Iterative design and sim-to-real." In: *Conference on Robot Learning*. PMLR. 2020, pp. 317–329.

[210] Bin Yan, Yi Jiang, Jiannan Wu, Dong Wang, Ping Luo, Zehuan Yuan, and Huchuan Lu. "Universal instance perception as object discovery and retrieval." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 15325–15336.

[211] Tianyu Yang and Antoni B Chan. "Learning dynamic memory networks for object tracking." In: *Proceedings of the European conference on computer vision*. 2018, pp. 152–167.

[212] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. "Improving sample efficiency in model-free reinforcement learning from images." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 10674–10681.

[213] Xiaopeng Yu, Jiechuan Jiang, Wanpeng Zhang, Haobin Jiang, and Zongqing Lu. "Model-based opponent modeling." In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 28208–28221.

[214] Amy Zhang, Harsh Satija, and Joelle Pineau. "Decoupling dynamics and reward for transfer learning." In: (2018).

[215] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. "Towards vision-based deep reinforcement learning for robotic motion control." In: *arXiv preprint arXiv:1511.03791* (2015).

[216] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. "Multi-agent reinforcement learning: A selective overview of theories and algorithms." In: *Handbook of Reinforcement Learning and Control* (2021), pp. 321–384.

[217] Qizhen Zhang, Chris Lu, Animesh Garg, and Jakob Foerster. "Centralized model and exploration policy for multi-agent RL." In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 2022, pp. 1500–1508.

[218] Tianjun Zhang, Michael Janner, Yueying Li, Tim Rocktäschel, Edward Grefenstette, Yuandong Tian, et al. "Efficient planning in a compact latent action space." In: *The Eleventh International Conference on Learning Representations*. 2022.

[219] Wenxuan Zhou, Sujay Bajracharya, and David Held. "PLAS: Latent action space for offline reinforcement learning." In: *Conference on Robot Learning*. 2020.

[220] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. "robosuite: A modular simulation framework and benchmark for robot learning." In: *arXiv preprint arXiv:2009.12293*. 2020.

[221] Zheng Zhu, Qiang Wang, Bo Li, Wei Wu, Junjie Yan, and Weiming Hu. "Distractor-aware siamese networks for visual object tracking." In: *Proceedings of the European conference on computer vision*. 2018, pp. 101–117.

[222] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.