UNIVERSITÀ DEGLI STUDI
DI GENOVA

ISTITUTO ITALIANO
DI TECNOLOGIA

Indirect Methods for Robot Skill Learning

by

Domingo Francisco Esteban Cabala

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

July 2019

Advisors:
Prof. Darwin G. Caldwell
Dr. Leonel Rozo

Dibris
Department of Informatics, Bioengineering, Robotics
and Systems Engineering

ISTITUTO ITALIANO
DI TECNOLOGIA
ADVANCED ROBOTICS

Abstract

Indirect Methods for Robot Skill Learning

Domingo Esteban

Robot learning algorithms are appealing alternatives for acquiring rational robotic behaviors from data collected during the execution of tasks. Furthermore, most robot learning techniques are stated as isolated stages and focused on directly obtaining rational policies as a result of optimizing only performance measures of single tasks. However, formulating robotic skill acquisition processes in such a way have some disadvantages. For example, if the same skill has to be learned by different robots, independent learning processes should be carried out for acquiring exclusive policies for each robot. Similarly, if a robot has to learn diverse skills, the robot should acquire the policy for each task in separate learning processes, in a sequential order and commonly starting from scratch. In the same way, formulating the learning process in terms of only the performance measure, makes robots to unintentionally avoid situations that should not be repeated, but without any mechanism that captures the necessity of not repeating those wrong behaviors.

In contrast, humans and other animals exploit their experience not only for improving the performance of the task they are currently executing, but for constructing indirectly multiple models to help them with that particular task and to generalize to new problems. Accordingly, the models and algorithms proposed in this thesis seek to be more data efficient and extract more information from the interaction data that is collected either from expert's demonstrations or the robot's own experience. The first approach encodes robotic skills with shared latent variable models, obtaining latent representations that can be transferred from one robot to others, therefore avoiding to learn the same task from scratch. The second approach learns complex rational policies by representing them as hierarchical models that can perform multiple concurrent tasks, and whose components are learned in the same learning process, instead of separate processes. Finally, the third approach uses the interaction data for learning two alternative and antagonistic policies that capture what to and not to do, and which influence the learning process in addition to the performance measure defined for the task.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Domingo Francisco Esteban Cabala
July 2019

Acknowledgements

First and foremost, I would like to thank my tutors Leonel Rozo and Darwin Caldwell for their support and guidance throughout my PhD. Leonel Rozo has not been only a good supervisor who gave me the freedom to explore and work on my ideas, but also a good friend who always started a conversion asking about how I was. Prof. Caldwell not only gave me the opportunity to perform research at IIT but also, despite his busy schedule, alway had time to listen to me.

I also want to thank Prof. Ville Kyrki and Dr. Bojan Nemec for reviewing this thesis, and providing helpful feedback that allowed me to improve this manuscript.

Additionally, I am very grateful to all the members of ADVR, past and present. I have learned so much from all of them during these years, they have been a key part of my academic and personal growth.

Finally, I would like to thank my family. Especially, my mother and brother for everything they have done for me and for always believing in me. Thanks to Lía for her patience and unconditional support throughout this journey. *Esta tesis se la dedico a ustedes. Gracias por todo, los amo*.

Contents

List of Figures

List of Tables

Acronyms

IL	Imitation Learning
BC	Behavioral Cloning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
GPS	Guided Policy Search
MDGPS	Mirror Descent Guided Policy Search
SAC	Soft Actor-Critic
HIU	Hierarchical Intentional-Unintentional
GP	Gaussian Process
GP-LVM	Gaussian process latent variable model
NN	(Artificial) Neural network
TVLG	Time-varying linear-Gaussian (controller)
KL	Kullback-Leibler (divergence)

Notation

$\mathcal{M}$	Markov decision process
$\mathcal{T}$	Task
t	Discrete time step
T	Horizon, final time step of an episode
$\mathbf{s}$	State
$\mathcal{S}$	State space
$\mathbf{s}_t$	State at time t
$\mathbf{a}$	Action
$\mathcal{A}$	Action space
$\mathbf{a}_t$	Action at time t
$p(\mathbf{s}_{t+1}\|\mathbf{s}_t,\mathbf{a}_t)$	Probability of transition to state $\mathbf{s}_{t+1}$, from state $\mathbf{s}_t$ after action $\mathbf{a}_t$
τ	Trajectory, path or rollout, $\tau = \{\mathbf{s}_0,\mathbf{a}_0,\mathbf{s}_1,\mathbf{a}_1,\dots\}$
r	Reward
r_t	Reward at time t
G_t	Return following time t
π	Policy
π^*	Optimal policy
$\pi(\mathbf{s})$	Deterministic policy
$\pi(\mathbf{a}\|\mathbf{s})$	Stochastic Policy
$\pi_{\boldsymbol{\theta}}$	Parameterized Policy
$V^{\pi}(\mathbf{s})$	Value of state $\mathbf{s}$ under policy π
$V^{*}(\mathbf{s})$	Value of state $\mathbf{s}$ under π^*
$V_{\boldsymbol{\psi}}(\mathbf{s})$	Parameterized value function of a state
$Q^{\pi}(\mathbf{s},\mathbf{a})$	Value of taking action $\mathbf{a}$ in state $\mathbf{s}$ under policy π
$Q^{*}(\mathbf{s},\mathbf{a})$	Value of taking action $\mathbf{a}$ in state $\mathbf{s}$ under π^*
$Q_{\boldsymbol{\phi}}(\mathbf{s},\mathbf{a})$	Parameterized value of a state-action pair

Chapter 1

INTRODUCTION

One of the greatest ambitions of humanity is and has been to build intelligent machines. The idea of making intelligent interactions between mechatronic systems, so-called *robots*, and the real-world, originated the science and technology that we currently know as robotics. Nevertheless, the classical procedure that has guided the development of robotics since its origins has been to identify the specific tasks that robots should execute and preprogram the required behaviors. This manual ad-hoc design presents some limitations. First, it biases the robotic behavior to the empirical and subjective idea of what the programmer considers as intelligent. And second, it is practically impossible to define in advance the complexity and variety of tasks that robots may face in the dynamic and unstructured real-world.

In accordance with current research, this thesis connects intelligence with the concept of rationality [1, 2]. This involves that a robot behavior can be evaluated with a *performance measure* that captures the objective of the task that the robot performs. Thus, the behavior should be optimal, that is rational, with respect to this objective during the whole execution of the task. This problem may be formalized as a *sequential decision-making* or dynamic optimization problem [3] by defining the variables that are relevant for the decision, so-called *state*, and the decision variables as control actions or simply *actions* of the robot. When the robot performs the sequence of actions obtained from the solution of this problem, the resulted behavior is rational with respect to the performance measure.

The sequential decision-making problem involves an interaction over time and, therefore, requires a transition model that describes how the state of the environment evolves with the actions performed by the robot. There are several tasks in robotics where it is possible to manually generate a model that describes the evolution of the state under the influence of the control actions. With this model, usually constructed from human insights of physics, a *policy* or control law that generates the motor commands based on the state of the task can be obtained with *optimal control* [4]. Optimal control ensures strong guarantees when the model perfectly captures the state-action relationship, however fails when some factors that affects the evolution of the task are not considered [5]. Additionally, there are several tasks that are difficult, if not impossible, to construct a perfect model which the optimal control techniques can work with.

Moreover, the generation of a robotic behavior with optimal control assumes the task environment is fully observable, that is, the robot has a full and reliable access to those variables relevant to the action selection. However, most robotic scenarios are partially observable because the robot's sensors does not give access to the complete state, and require the robot to estimate both its own internal state (proprioception) and that of the external world (exteroception) from its sensor measurements [1, 6]. This perception process requires the robot to continuously update its belief about the environment state (so-called belief state), based on the sequence of previous actions and percepts. However, obtaining a reliable state by robotic perception is difficult because sensors are noisy, and the real-world is unpredictable, dynamic and partially observable.

1.1 Learning motor skills in robotics

The lack of a perfect model of the state-action correlation or a full access to the state of the environment have not been a limitation for the development of the astonishing motor skills that humans and other animals possess. The control laws that generate these behaviors are consequence of the habituation and experience they obtain from interacting with their environment. Similarly, an alternative approach for acquiring motor skills in robotics is to allow robots to practice the desired task and improve their performance on that task with experience, that is learning [7, 8].

Robot learning of motor skills is the scientific study of machine learning techniques to automatically detect uncovered patterns from the data obtained from the execution of one task, in order to acquire a control policy that generates a rational behavior with respect to the performance measure defined for that task. The data generation process employed to collect the required experience defines the appropriate robot learning approach for improving the performance of the robot's policy [5]. For example, when the task execution is provided by an expert or demonstrator, the robot can learn the desired behavior by emulating it, an approach called *Imitation Learning (IL)*, also called apprenticeship learning or programming by demonstration [9]. On the other hand, when the interaction data is collected following the robot's policy, the skill is improved by *Reinforcement Learning (RL)* with the performance measure defined for the task [10].

1.2 Limitations of direct learning processes

There has been a great deal of work on the development of learning algorithms for the acquisition of newly robot skills via either imitation learning or reinforcement learning [11, 12, 13, 14]; however, we are still far from having robots with the sufficient degree of autonomy to operate continuously in the real-world. The ultimate goal of robot learning is to obtain a rational policy since it is sufficient to generate the robot's behavior. Thus, most robot learning techniques are only focused on obtaining policies as a result of optimizing the performance measure of the task. Nevertheless, formulating the skill acquisition process in such a way involves deriving the policy directly from the solution of this optimization problem, which has several disadvantages. For example,

- if the same skill has to be learned by different robots, independent learning processes should be carried out for acquiring the policy for each robot. That is to say, after a — probably long and cumbersome — learning process, the resulted policy of one robot is only valid for that particular robot and that specific task, and does not generalize to others easily.
- Because the interaction data is only used for improving the performance in a single task, if a robot has to learn diverse skills, the robot should acquire the policy for each task in separate learning processes, in a sequential order and commonly starting from scratch.
- Because the learning process is formulated in terms of improving only the performance measure, the robot may ignore situations that should not be repeated (failed interactions/behaviors). These failures are only indirectly avoided by reducing the likelihood of occurrence, but without any mechanism that captures the necessity of not repeating those behaviors.

In contrast, humans and other animals exploit their experience not only for improving the performance of the task they are currently executing, but for constructing multiple complex models to help them with that task and to generalize to new problems. Therefore, robot learning algorithms should not be designed to consider the skill acquisition process as an isolated stage where the only objective and final result is a policy with acceptable performance in the task being executed. Instead, the algorithms should consider the skill acquisition process as a part of a higher continual learning activity where the experience collected during the task execution is reused and integrated into an overall intelligent system [15].

In consequence, robot learning algorithms should be more data efficient and extract more information from the interaction data that is collected either from expert's demonstrations or the robot's own experience [16, 17]. Thus, this interaction data should not be employed solely for acquiring a robot's policy in one particular task, but also for learning alternative and complementary models that support the robot in the current and following tasks. Three models formulated in these type of scenarios and algorithms that learn their different components in the same learning process are briefly detailed in the following section.

1.3 Contributions and thesis outline

The algorithms proposed in this thesis intend to overcome the problems detailed in the previous section by learning the robotic motor skills indirectly. That is to say, the rational policy is not obtained directly from an optimization problem which is formulated only in terms of the performance measure. Instead, these algorithms seek to extract additional information from the experience obtained from the task execution and construct models that are useful for both the current learning process and future ones. Therefore, the policy is derived from alternative optimization objectives that influence it during the learning process. But before introducing these algorithms, a mathematical formulation of the motor skill learning problem, and the definition of some important concepts are presented in chapter 2.

In chapter 3, it is shown the benefit of explaining the evolution of environment states and robot actions, with variables in a shared latent space, called skill space. With this model, the transition of these latent variables are enough to generate the control commands of the robot. As a consequence, when the same skill has to be acquired by different robots, and under the assumptions that the state and the influence of an action to the evolution of this state are both independent of the robot executing the task, the latent representation of one robot can be used to learn the latent representation for others. This process for *learning a shared latent space of a robotic skill and transfer it to other robots* is depicted in Figure 1.1. By transferring this prior latent knowledge from one robot to another, a faster learning rate along with good imitation performance are achieved. The skill space is modeled with a Gaussian process latent variable model (GP-LVM) [18], a model previously applied in the context of imitation of human actions [19, 20], and adapt it to this context of transfer learning. *Part of the content of this chapter was published previously in [21].*

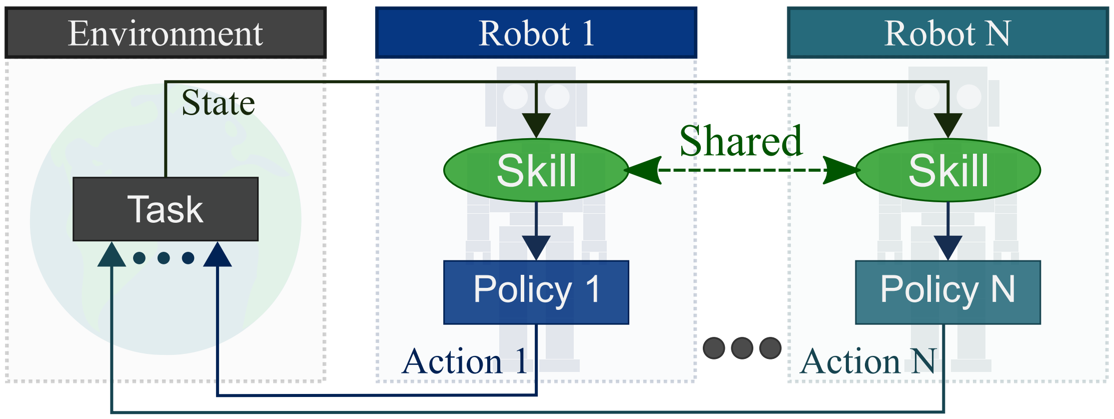


Figure 1.1 *Shared latent spaces of robotic skills.* The same skill has to be learned by different robots (chapter 3).

In chapter 4, it is shown how complex learning tasks can be decomposed into collections of subtasks that are simpler to learn as well as reusable for new problems. Moreover, the corresponding policies are modeled as conditional Gaussian distributions and combined using a set of activation vectors. The result of this approach is a compound Gaussian policy that is a function of the means and covariances matrices of the composable policies. This model allows performing tasks that require the concurrent accomplishment of several subtasks, compared to standard models that consider only sequential subtasks. In addition, the chapter details an algorithm, called Hierarchical Intentional-Unintentional (HIU), which exploits the off-policy data generated from the compound policy for *learning both compound and composable policies within the same learning process*, as depicted in Figure 1.2. The algorithm is built on a maximum entropy reinforcement learning approach to favor exploration during the learning process. The results obtained from the conducted experiments suggest that the experience collected with the compound policy permits not only to solve the complex task but also to obtain useful composable policies that successfully perform in their respective tasks. *This work has been disseminated in [22].*

In chapter 5 it is shown that the interactions that result in failures, and that are usually overlooked for the robot's policy, can be explicitly considered if their effects are captured with a policy that can reproduce it, called bad policy. At the same time, a good policy can be constructed from only successful executions, and along the bad policy, may influence the skill acquisition process. This process of *learning a good and bad policy from the interaction data and make them influence the robot's policy during the learning procedure* is depicted in Figure 1.3. However, instead of considering them directly in the policy improvement

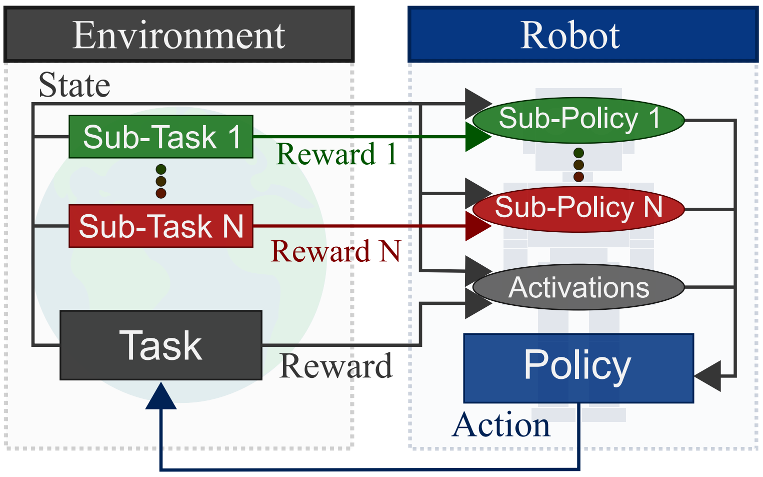


Figure 1.2 *Simultaneous learning and composition of modular policies.* Different policies should be simultaneously learned by the same robot and composed for solving a complex task (chapter 4).

phase of deep neural network policies, a guided policy search algorithm [23] is extended so it can consider guiding trajectories optimized with dualist constraints [24]. These constraints are aimed at assisting the policy learning so that the trajectory distributions updated at each iteration are similar to trajectory distributions generated by good policies while differing from trajectory distributions generated from bad policies. The results indicate that neural network policies guided by trajectories optimized with this method reduce the failures during the policy exploration phase, and therefore encourage safer interactions. *This work was published previously as [25].*

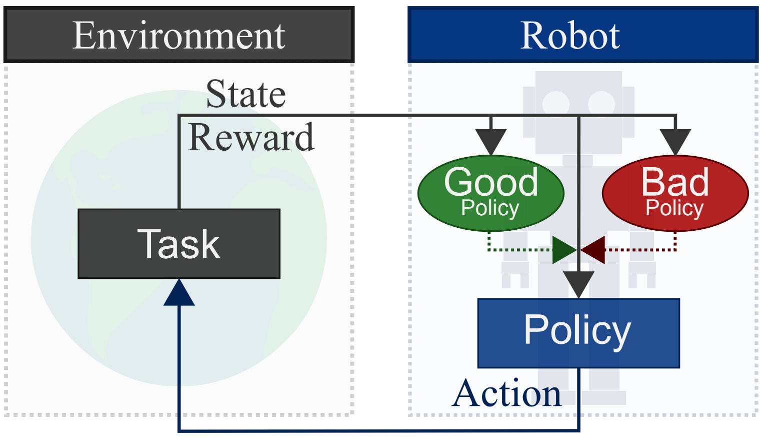


Figure 1.3 *Exploiting good and bad experiences in policy learning.* Different policies are learned from the same experience and all influence the learning process of one robot (chapter 5).

Finally, in chapter 6, the conclusions of this thesis are presented and the potential avenues for future research discussed.

Chapter 2

BACKGROUND

This chapter presents the mathematical formulation of the learning problem for robotic skill acquisition considered in this thesis. The solution to this problem is a control policy that generates a rational behavior with respect to the performance measure. As mentioned in the previous chapter, the origin and type of data obtained during the execution of the robotic task determines the approach that could be used to learn the policy. When this interaction data is generated by an expert, the policy acquisition can be approached as an imitation learning problem. This approach is described in section 2.1 and shows how the skill learning problem is reduced to supervised learning, and then it allows to exploit the robust machine learning algorithms offered in this kind of problems. On the other hand, when the robot has only access to data obtained directly from its interaction with the environment, the policy is updated in such a way that improves the performance measure. Thus, the problem could be approached with the reinforcement learning framework (section 2.2), and the rational behavior is obtained in a manner analogous to trial and error.

Let us assume that a robot has to develop a particular skill for the *task* $\mathcal{T}$ and that the performance of the resulted behavior can be measured. This performance measure, also called utility function, is a mathematical expression of what is considered a desirable or rational behavior [1]. Consequently, all the decisions taken by the robot during the whole task influence the resulted behavior and should be considered in the assessment. In this thesis two important assumptions are made about the characteristics of this decision process. First, a discrete-time setting is considered, and hence the robot's decisions are made at discrete instances of time or stages. And second, the environment where the robot performs the task

[1]This idea is also known as the *reward hypothesis* which states that the goals and purposes we expect from an agent can be formulated as the maximization of the performance measure [26].

is considered stochastic in order to reflect the stochasticity of the real-world [2]. The previous assumptions describe a discrete-time stochastic sequential decision (control) problem which could be formalized as a *Markov Decision Process* (MDP).

An MDP is described by the tuple $(\mathcal{S}, \mathcal{A}, p_s, r)$. The *state space* $\mathcal{S}$ has dimension $\mathrm{D}_{\mathcal{S}}$, and the element $\mathbf{s} \in \mathcal{S}$ is called *state*. The *action space* $\mathcal{A}$ has dimension $\mathrm{D}_{\mathcal{A}}$, and the element $\mathbf{a} \in \mathcal{A}$ is called control *action*. The *transition function* is represented by $p_s : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ and defines the conditional probability distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, that is the probability to transition to $\mathbf{s}_{t+1}$, from state $\mathbf{s}_t$ after action $\mathbf{a}_t$. Finally, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the *reward function* that assesses the state-action pair by a scalar value, $r(\mathbf{s}, \mathbf{a})$. [3]

Some assumptions about the characteristics of the MDPs formulated in this thesis are made in order to consider robotic tasks. First, both $\mathcal{S}$ and $\mathcal{A}$ are (possibly high dimensional) continuous spaces, therefore $\mathcal{S} \subset \mathbb{R}^{\mathrm{D}_{\mathcal{S}}}$ and $\mathcal{A} \subset \mathbb{R}^{\mathrm{D}_{\mathcal{A}}}$. This also implies that the transition $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is now a probability density function $p_s : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, \infty)$. Moreover, it is considered that the robot can directly observe the real state of the MDP (fully observable MDP), and that the state is composed by either internal (e.g. joint angles, end-effector pose) or external variables (e.g. object locations).

The robot actions $\mathbf{a}$ are selected according to a control *policy*, denoted by π, that can be either deterministic $\pi(\mathbf{s})$ or stochastic $\pi(\mathbf{a}|\mathbf{s})$, being the latter a distribution over actions $\mathbf{a}$ conditioned on the state $\mathbf{s}$. With this policy, the robot induces a stochastic process called *trajectory*, path, or rollout, $\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots\}$. In the following we will denote the probability of the trajectory induced for the policy π as $\pi(\tau)$.

Le us also define the *return* G_t, as the accumulated reward obtained at each time step, $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$, for a particular trajectory starting at time t. Moreover, the return for a trajectory starting at $t = 0$ is denoted by $G(\tau)$. Thus, when the task $\mathcal{T}$ has to be performed in a fixed time horizon T, we called it *episodic task*, and the resulting trajectory is assessed by the *finite-horizon undiscounted return* $G(\tau) = \sum_{t=0}^{T-1} r_t$. In some special cases, the task does not have a horizon, so-called *continuing task*, and the trajectory is assessed by the *infinite-horizon discounted return* $G(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$, where the *discount rate* $\gamma \in [0,1)$, allows the infinite sum to converge. In both cases, the skill performance of the robot in the task $\mathcal{T}$ is measured by the *expected return*, $J(\pi)$:

[2] This stochastic assumption seeks to capture the complexity, uncertainty and the lack of information that we have about the real-world.

[3] Other formulations of the reward function, such as $r_t = r(\mathbf{s}_t)$ or $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, are also possible. However, the decision for choosing $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$ is mainly justified for its convenience in the formulation and code implementation of the algorithms proposed in this thesis.

$$J(\pi) = \int_{\tau} \pi(\tau) G(\tau) = \mathbb{E}_{\tau \sim \pi(\cdot)}[G(\tau)] \tag{2.1}$$

The policy that maximizes the expected return is called *optimal policy*, π^*, and it is obtained by solving the problem:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} J(\pi) \tag{2.2}$$

Because both $\mathcal{S}$ and $\mathcal{A}$ are continuous, an exact solution to the MDP is not possible. Nevertheless, it is possible to find an approximation of π^* by using a *parameterized policy* $\pi_{\boldsymbol{\theta}}$, with parameters $\boldsymbol{\theta}$. The optimal parameter values, $\boldsymbol{\theta}^*$, of this model are obtained by solving:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} J(\pi_{\boldsymbol{\theta}}) \tag{2.3}$$

We say that the robot is *learning*, when the value of $J(\pi_{\boldsymbol{\theta}})$ improves with the experience obtained during the execution of task $\mathcal{T}$. The data required for learning this skill can come either by an expert executing the task (section 2.1) or from the robot own experience (section 2.2).

2.1 Robot learning from observed expert data

An alternative to learning a robotic skill from interaction data collected from the robot's experience is exploiting the task-specific knowledge of an expert or demonstrator about how to perform the task [9]. When the task execution is provided by the demonstrator, the robot has the interaction data from which it can learn the desired behavior by emulating it, an approach called *Imitation Learning* (IL), also known as apprenticeship learning or programming by demonstration [9, 27, 28]. The demonstrations can be provided by several interfaces, such as recorded human motions [29, 30], kinesthetic teaching [31, 32], or teleoperation [33, 34]. It is common that the demonstrator provides a certain number of executions of the task to capture variability in the task execution, something that is helpful for obtaining distributions from the demonstrations. However, it is still possible to learn from a single demonstration, a process called one-shot imitation learning [35, 36].

There are two main approaches that can be followed in imitation learning to capture the expert knowledge about the task. The first approach is to assume that the expert acts (near) optimally with respect to some performance measure, and then try to learn the reward function that explains the experience data. Then from this reward function, the expert policy is derived. This process is called, *inverse reinforcement learning* or *inverse optimal control* [12].

The second approach is to obtain directly the expert policy from the data, this approach is called *behavioral cloning*. Behavioral cloning (BC) is the more robust approach in robot learning, and the variety of successful tasks learned with this method is broad in the bibliography [27].

2.1.1 Imitation learning by behavioral cloning

Let us define a *demonstration*, $\tau^{\mathrm{E}} = \{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\}$, as a trajectory performed by an *expert*, also called demonstrator, in the task $\mathcal{T}$ for a time horizon T. Assuming that D demonstrations are collected in a dataset $\mathcal{D} = \{\tau_1^{\mathrm{E}}, \ldots, \tau_D^{\mathrm{E}}\}$, the objective of imitation learning is to learn a parameterized policy $\pi_{\boldsymbol{\theta}}$ that can reproduce the expert skill, π^{E}, observed in $\mathcal{D}$. As we can notice, the dataset $\mathcal{D}$ consists of a collection of expert actions taken for different states. This input-output problem is well studied in machine learning, therefore, the policy $\pi_{\boldsymbol{\theta}}$ might be learned by supervised learning. As a result, learning the parameters $\boldsymbol{\theta}$ by behavioral cloning is usually done by optimizing a regression loss function.

A common way to estimate the policy parameters is by maximum likelihood estimation (MLE) [37], if we assume that the state-action pairs $(\mathbf{s}, \mathbf{a})$ of the data are independent and identically distributed (i.i.d.). Thus, the optimal policy parameters $\boldsymbol{\theta}^*$ can be obtained by maximizing the likelihood of the demonstrations with $\pi_{\boldsymbol{\theta}}$:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_n | \mathbf{s}_n)$$

where $N = DT$ is the total number of state-action pairs in the dataset D. Therefore, the optimal policy learned by BC is the one that maximizes the likelihood of the expert trajectories.

It is important to note that assuming the state-action pairs are i.i.d. is a very strong assumption. As described earlier, the values of the states depend on previous actions, then the assumption is easily violated. However, policies modeled with general function approximators can still reproduce the expert skill if the dynamics of the task is deterministic (or very close to) or if the dataset $\mathcal{D}$ is big and representative.

2.2 Robot learning from the experienced interaction data

Learning a skill by imitation learning may be beneficial in certain scenarios but it has several limitations. Obtaining demonstrations that are both correct and representative of all the situations a robot might encounter is difficult or even unrealistic in some situations. An alternative and general approach is to allow the robot to autonomously interact with the environment and improve its policy directly from this experience. Reinforcement learning (RL) is a subfield in machine learning that deals with problems where the agent has not access to the transition function, that is an incompletely-known MDP.

With RL, the robot acquires a skill in task $\mathcal{T}$ by means of the reward signal. In addition to the delayed reward feature of sequential decision making problems, in the RL framework, the robot must discover which actions yield the most reward by trying them, in other words it should perform a trial-and-error search. As a consequence, RL presents an exploration-exploitation dilemma for the robot, because it has to exploit what it has already experienced in order to obtain a better performance, but it has also to explore in order to select better actions in the future [26]. RL algorithms can be classified according to diverse criteria, however, in the following sections two main classes, model-free (also called direct) and model-based (also called indirect) methods are briefly introduced.

2.2.1 Model-free reinforcement learning

The main difference between model-free and model-based RL methods is whether a model of the interactions, that is a transition function, is learned and exploited. Model-free RL algorithms do not learn this model, as a result the optimal actions are obtained directly from trial-and-error with the physical world. Several RL algorithms estimate *value functions* which relates either states or state-action pairs to the expected return under some specific policy π. Thus, given a state $\mathbf{s}$ the *state-value function* for π is denoted by $V^{\pi}(\mathbf{s})$ and defined as:

$$V^{\pi}(\mathbf{s}) = \mathbb{E}_{\pi}\left[G_t \mid \mathbf{s}_t = \mathbf{s}\right] \tag{2.4}$$

On the other hand, given a state $\mathbf{s}$ and an action $\mathbf{a}$, the *action-value* function for policy π is denoted by $Q^{\pi}(\mathbf{s},\mathbf{a})$ defined as:

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = \mathbb{E}_{\pi}\left[G_t \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right]. \tag{2.5}$$

An important property of these value functions is that they can be expressed in terms of the successive states, recursive expressions called *Bellman equations*. For example, considering an infinite-horizon discounted return, the Bellman equation for the action value function is:

$$Q^{\pi}(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{(\mathbf{s}',\mathbf{a}')\sim\pi}\left[Q^{\pi}(\mathbf{s}',\mathbf{a}')\right], \tag{2.6}$$

where $\mathbf{s}'$ and $\mathbf{a}'$ denote the next time-step state and actions.

The *optimal action-value function* and *optimal state-value function*, denoted by V^* and Q^* respectively, are defined as:

$$V^*(\mathbf{s}) = \max_{\pi} V^{\pi}(\mathbf{s}), \quad \forall \mathbf{s} \in \mathcal{S} \tag{2.7}$$

$$Q^*(\mathbf{s},\mathbf{a}) = \max_{\pi} Q^{\pi}(\mathbf{s},\mathbf{a}), \quad \forall \mathbf{s} \in \mathcal{S}, \forall \mathbf{a} \in \mathcal{A} \tag{2.8}$$

Similarly to (2.6), the optimal value functions can be expressed recursively, formulations called *Bellman optimality equations*. Thus, the Bellman optimality equation for Q^* is:

$$Q^*(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \mathbb{E}_{\mathbf{s}'}\left[\max_{\mathbf{a}'} Q^*(\mathbf{s}',\mathbf{a}')\right]. \tag{2.9}$$

These optimal value functions assign to each state, or state-action pair, the largest expected return achievable by any policy. For this reason, the optimal policy π^* can be recovered directly from the optimal action-value function, without learn it explicitly. The RL algorithms based on this idea are commonly named *value-based methods*.

Most model-free value-based methods are grounded on temporal difference (TD) learning, where on every time-step the agent interacts with the environment and updates its value function estimate based on previous learned estimates (bootstrapping). The classical TD methods SARSA [38] and Q-learning [39], which learn estimations of the Q-values for the exploration policy and optimal Q-values respectively, require an exact representation of the Q-function, e.g. a table indexed by discrete states and actions. Thus, function approximation is required in order to deal with value functions of the continuous spaces found in robotics [40, 41]. In this thesis, a parameterized approximator of the Q-values is denoted by $Q_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\phi}$. Learning $Q_{\boldsymbol{\phi}}$ with interaction data allows to generalize and estimate values to states and actions not experienced by the robot. However, several tasks still require an intensive exploration of the state-action space in order to learn complex value functions. Similarly, learning value approximations is difficult in high-dimensional spaces and errors in the value functions approximations are directly propagated through to the policy that is implicitly represented [42].

On the other hand, an alternative collection of RL algorithms learn the optimal policy directly based on sampled trajectories, and avoid computing any optimal value function. They are called *policy search* or policy-based methods and optimize the parameters $\boldsymbol{\theta}$ of the parameterized policy $\pi_{\boldsymbol{\theta}}$ in order to optimize the performance measure of the task (Equation (2.3)). Most policy search methods follow a process that iterates over the following steps: *exploration*, that is generating samples with the current policy; policy *evaluation*, that is evaluate the policy performance; and policy *improvement* where the policy parameters are updated [42]. Several policy search algorithms estimate the policy performance with Monte Carlo sampling, that is running a large set of episodes and averaging over the stochastic return of the resulting trajectories [26]. For this reason, these algorithms are also known as transient critic methods [17], because the estimations of the policy performance are discarded once the policy parameters are updated.

Monte Carlo estimates are unbiased however they exhibit a high variance that can produce slow learning [42, 43]. As a consequence, actor-critic methods learn the parameterized policy (so-called actor) that maximizes a parameterized value-function (so-called critic) instead of the Monte Carlo estimations [26, 44]. The parameterized critic is learned with TD, for this reason actor-critic methods are also called persistent critic algorithms [17]. Since the parameterized critic is a low variance estimator of the expected return, actor-critic methods often converge much faster than policy search methods [45].

2.2.2 Model-based reinforcement learning

The interaction data obtained from experience can also be used to construct a model of how the actions affect the evolution of the states, in a process called *model learning*. RL methods that use these transitions models are commonly denoted as model-based RL methods [46]. The performance of the learned transition model is given by its generalization ability, that is how precise its predictions are on new state-action pairs. And they are used to simulate the environment allowing state-space planning in order to improve the performance of the policy [1, 26].

From a function approximation point of view, model learning methods are classified as global and local techniques [47]. Global regression techniques model the transition function using all the experienced data, as a consequence every state-action pair influences every parameter of the transition model. Relevant models exploited with global techniques include parametric ones such as neural networks [48, 49], and non-parametric ones such as Gaussian processes [16, 50]. On the other hand, local regression techniques estimate the transition function around a query state-action point, and the experienced interaction data close to the query point is used to predict the next state. As a consequence, the predictions are more accurate for query points similar to previously experienced interactions, otherwise there is not guarantee in the validity of the predictions. Local regression methods are simple to implement and computationally efficient, therefore both linear [23, 51, 52], and nonlinear models [53, 54] are used in robotics with this type of methods.

Chapter 3

LEARNING AND TRANSFERRING SHARED LATENT SPACES OF ROBOTIC SKILLS

Obtaining a rational policy is the ultimate goal in robot learning for control since it is sufficient to generate the robot's behavior and thus for a robot to be skilled in a particular task. This policy maps perceived states to actions (or probabilities over actions) and therefore it is specific for certain task and robot. Two rational policies for distinct tasks are different because they optimize their own performance measure, and thus they have different criteria for selecting the actions. On the other hand, two robots may even have the same sensors and actuators, however, minor differences among them might cause them to perceive the state or modify the environment differently. As a consequence, applying the same policy in the two robots may lead to two distinct resulting behaviors. Thus, acquiring the same skill for two particular robots involve learning two different policies.

With common robot learning methods, training two different policies imply to carry out two independent learning processes. However, when two robots exhibit the same behavior in the same task, one might expect some commonalities in the decision criteria they consider to choose their particular actions. As a consequence, the state-action trajectories they induce when they use their policies should also have similarities during the task execution. Therefore, learning the cause of a skill means finding these invariant patterns in the interaction data obtained for each robot.

The previous idea considers that the interaction data collected among the different robots is independent of the generation process that was used to collect it. However, in this chapter the proposed algorithm, and the assumptions made, are formulated in a behavioral cloning

(BC) setting. The main motivation is that the success of a skill acquired by BC depends heavily on the demonstrations. Therefore, learning a robotic skill requires to collect a large and diverse dataset, a process that may become laborious when dealing with high-DoF robots, for example. The data collection is even more tiresome if the same skill needs to be learned by multiple robots, because the demonstrations have to be done in such a way that allow obtaining a similar behavior in all the robots. In this context, learning the causes that generate the different skills is valuable because the demonstrator can first train one robot and exploit this knowledge to train others, accelerating the overall training process. This scenario matches the paradigm of transfer learning, an approach where prior (learned) information is exploited in subsequent learning phases [55].

The demonstrated state-action trajectories reflect the skill the demonstrator wants the robot to learn. In this context, it is possible to assume that the demonstrations from this particular skill can be generated by variables in a latent space that explains the patterns found in both state and action data. If we also learn the map from the state to this latent space, then the policy we seek to learn with BC is a state-latent-action mapping [56]. Furthermore, when the state representation of the environment and the influence of an action to the evolution of this state are both independent of the robot executing the task, then the expected state trajectory collected with the same skill should be the same among the robots. For example, if the state of an object is represented by its position and velocity, and under the same external conditions, any robot that moves this object to some point with a specific acceleration, will generate the same state trajectory. Clearly, this situation is not necessarily true for the action trajectories of the robots, because the actions required to generate the state trajectory are particular to each robot.

Under the previous assumptions, the mapping from the state to this latent space is the same among all the robots, hence the latent representation of one robot (source domain) can be used to learn the latent representation for others (target domains). In other words, teaching the same skill among many robots can be carried out by first learning a latent representation of the demonstrations for one robot, and then exploiting it for training other ones. By transferring this prior latent knowledge from one robot to another, we expect to achieve a faster learning rate along with good imitation performance.

In this chapter, a shared GP-LVM [18] is proposed for modeling the shared latent space. GP-LVM is a data-efficient method that allows a nonlinear mapping from a latent space to observational spaces. Moreover, when BC is used for teaching skills to complex robots, like humanoids, the dimensionality of the training data can considerably increase, making both imitation and transfer learning more complex processes. In this context, despite the dimension of the latent space modeled with a GP-LVM can even be higher than observations and action spaces [19], it is possible to overcome the curse of dimensionality, by assuming that both state and action data lie in a shared low dimensional latent space. The details about how shared GP-LVM is formulated under this context is detailed in section 3.3.3.

3.1 Related work

Unsupervised learning is a branch in machine learning focused on discovering structure or patterns in the training data, a process that is also called knowledge discovery [37]. A relevant application of unsupervised learning is to build density models that can generate the training data, also-called generative models. Among them, a latent variable model (LVM) is a statistical model that assumes the training data (observed variables) is correlated because it is generated from a hidden common "cause" [37]. An important application of LVMs in robotics is dimensionality reduction in learning and control [57]. For example, reducing the state space to accelerate learning with linear methods such as Principal Component Analysis (PCA) [58, 59], or non-linear methods such as Gaussian process latent variable models (GP-LVM) [60].

Field et al. [61] proposed to learn a model of motion primitives for humanoid robots using hidden Markov models (HMM) and a mixture of Factor Analyzers (MFA). The main idea was to create a nonlinear low-dimensional mapping between human and robot using MFA, and reproduce the trajectories using an HMM-based dynamical system. The approach proposed here is similar to this approach in the sense that a shared demonstrator-robot low dimensional manifold is also found, however here the construction of the latent space and the mapping are wrapped in the same model, and therefore, their parameters are learned at the same time. Moreover, none of the aforementioned works addressed the problem of teaching skills to multiple robots. So, the proposed approach goes one step further by exploiting the learned latent space to deal not only with the curse of dimensionality and the human-robot mapping, but also with prior knowledge transfer — embedded in a latent representation of the motions — to quickly learn new mappings for multiple robots.

The idea of this chapter about exploiting shared GP-LVMs as an indirect input-output mapping was inspired by the work of Yamane et al. [19] in computer graphics and Shon et al. [20, 56] in robotics. Yamane et al. [19] applied a shared GP-LVM to animate characters with different morphologies, but a new model was learned for each character. Shon et al. [56] applied shared GP-LVM for imitation of human actions in a small humanoid. The shared latent variable model presented in this chapter extends on those works by considering multiple similar robots while differing in a major point: the input-to-latent mapping. In Yamane et al. [19], given new inputs, the latent coordinates were determined using a nearest neighbor search and an optimization process. The performance heavily depends on the number of nearest points to be chosen, which may be considerably high, limiting the use of this approach for real-time systems. To alleviate this problem, Shon et al. [20] applied a GP regression for the input-to-latent mapping once the model was trained. The shared GP-LVM proposed in this chapter includes a constraint in order to obtain a smooth input-to-latent mapping (so-called back-constraint), thus obtaining both the latent coordinates and the input-to-latent mapping variables during the optimization process. As explained in section 3.3.3, imposing back-constraints on the input presents several advantages, but more importantly, it is important for the transfer learning process.

The use of dimensionality reduction techniques in transfer learning has also been proposed in [62]. This approach finds a common low-dimensional latent space between source and target domains (for input variables), and then learns a latent-to-output mapping in the source domain. The transfer learning occurs when this mapping is applied to the target domain. This approach differs from the one proposed in this chapter in that the latent space of the latter is not only between input and output spaces, but also between source and target domains. Moreover, the approach in this chapter learns a new latent-to-output mapping in the target domain.

Transfer learning has also been applied to improve model learning in robotics. Nevertheless, these approaches do not make any distinction regarding the input and output variables of the problem, but they carry out a dimensionality reduction over the whole dataspace instead. By doing this, source and target low dimensional representations are different, and a linear [63] or a non-linear [64] transformation between them is required. In contrast, the shared latent variable model proposed in this chapter does not group inputs and outputs together to perform the dimensionality reduction. In addition, no transformation between the models is required because the source and target inputs coincide, and therefore the same latent coordinates and the hyperparameters of the state-to-latent mapping are assumed to be shared among the robots.

Finally, the approach presented in this chapter was previously published in Delhaisse and Esteban et al. [21]. Compared to that work, this chapter formalizes the main idea of transfer learning of robotic skills, and states some relevant assumptions.

3.2 Preliminaries

This section briefly describes Gaussian processes, a non-parametric model extensively used in the robot learning community for its data-efficiency and probabilistic properties. A complete review of Gaussian processes is outside the scope of this thesis, and a thorough presentation of this model can be found in Rasmussen and Williams [65] or Murphy [37]. However, the use of Gaussian process regression in a behavioral cloning setting is described in section 3.2.1. In addition, the unsupervised learning method that will be used to learn the latent space from a collection of expert demonstrations, so-called GP-LVM, is described in section 3.2.2.

3.2.1 Direct policy learning with Gaussian processes

Gaussian Processes (GPs) are non-parametric models that define probability distributions over functions in which any finite collection of the function values are assumed to be jointly Gaussian [65]. GPs are trained following a Bayesian approach by defining first a prior distribution about the characteristics of the functions we seek to obtain and then obtaining a distribution over functions to be consistent with the training data. GPs are common policy representations in robot learning due to their computational tractability for both inference and learning [66]. This section provides a brief overview of GP regression for modeling a parameterized policy $\pi_{\boldsymbol{\theta}}$ that imitates the expert policy that generated the demonstration data $\mathcal{D}$.

First, let us define the matrices $\mathbf{S} = [\mathbf{s}_1 \cdots \mathbf{s}_N]^\mathsf{T} \in \mathbb{R}^{N \times \mathrm{D}_\mathcal{S}}$ and $\mathbf{A} = [\mathbf{a}_1 \cdots \mathbf{a}_N]^\mathsf{T} \in \mathbb{R}^{N \times \mathrm{D}_\mathcal{A}}$ the state and action data respectively, where N is the number of training state-action pairs. Our beliefs about the characteristics of the plausible policy functions $\mathbf{f}$ that can produce our demonstrated actions can be represented by the GP prior $\mathbf{f} \sim \mathcal{GP}(\mathbf{0}, \mathbf{K}(\mathbf{S}, \mathbf{S}'))$, with zero mean and kernel matrix $\mathbf{K}$ that is the one that allow us to encode these beliefs [67]. For example, assuming that the sampled functions from this GP should be smooth, we can select the Squared Exponential (SE) kernel[1]. Extended with Automatic Relevance Determination (ARD) [65], each entry of this kernel matrix is given by

$$k(\mathbf{x}, \mathbf{x}') = \alpha^2 \exp\left(- \sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2 l_d} \right) \tag{3.1}$$

where D denotes the dimension of the data points $\mathbf{x}$ and $\mathbf{x}'$, and in which the amplitude α and the lengthscales l_d represent the policy parameters $\boldsymbol{\theta}$.

[1]also known as Radial Basis Function (RBF) kernel or Gaussian kernel.

By observing some pairs of data points $\{(\mathbf{s}_n, \mathbf{a}_n)\}_{n=1}^N$, we can update our beliefs about the distribution of policy functions. Assuming the likelihood is also Gaussian, that is $p(\mathbf{A}|\mathbf{F},\mathbf{S}) = \prod_{n=1}^N \mathcal{N}(\mathbf{a}_n|\mathbf{f}_n, \sigma^2\mathbf{I})$, where σ^2 represents the noise variance and $\mathbf{F} = [\mathbf{f}_1, ..., \mathbf{f}_N]^\mathsf{T}$, the marginal likelihood

$$p(\mathbf{A}|\mathbf{S};\boldsymbol{\theta}) = \int p(\mathbf{A}|\mathbf{F},\mathbf{S})p(\mathbf{F}|\mathbf{S};\boldsymbol{\theta})d\mathbf{F}, \tag{3.2}$$

can be calculated analytically. In this way, the optimal set of policy parameters $\boldsymbol{\theta}^*$ for the GP can be obtained by maximization of (3.2):

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} p(\mathbf{A}|\mathbf{S};\boldsymbol{\theta}). \tag{3.3}$$

Following the derivation in [67], we can predict an action $\mathbf{a}_i$ for a new state $\mathbf{s}_i$ from our policy by:

$$\mathbf{a}_i \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_i|\mathbf{s}_i) = p(\mathbf{a}_i \mid \mathbf{S},\mathbf{A},\mathbf{s}_i;\boldsymbol{\theta}) = \mathcal{N}(\mathbf{m}(\mathbf{a}_i), \mathbf{cov}(\mathbf{a}_i)) \tag{3.4}$$

with

$$\begin{aligned}\mathbf{m}(\mathbf{a}_i) =& \mathbf{K}(\mathbf{S},\mathbf{s}_i)^\mathsf{T}(\mathbf{K}(\mathbf{S},\mathbf{S}) + \sigma^2\mathbf{I})^{-1}\bar{\mathbf{a}} \\ \mathbf{cov}(\mathbf{a}_i) =& \mathbf{K}(\mathbf{s}_i,\mathbf{s}_i) - \mathbf{K}(\mathbf{S},\mathbf{s}_i)^\mathsf{T}(\mathbf{K}(\mathbf{S},\mathbf{S}) + \sigma^2\mathbf{I})^{-1}\mathbf{K}(\mathbf{S},\mathbf{s}_i),\end{aligned}$$

where $\mathbf{K}(\mathbf{S},\mathbf{s}_i)^\mathsf{T} \in \mathbb{R}^{\mathrm{D}_\mathcal{A} \times N\mathrm{D}_\mathcal{A}}$ has entries $(\mathbf{K}(\mathbf{s}_i,\mathbf{s}_j))_{d,d'}$ for $j = 1, \ldots, N$ and $d, d' = 1, \ldots, \mathrm{D}_\mathcal{A}$. The policy parameters $\boldsymbol{\theta}$ are the set of hyperparameterers of the kernel function $k(\mathbf{x},\mathbf{x}')$ used to compute $\mathbf{K}(\mathbf{S},\mathbf{S})$ and the variance σ^2.

Modeling a parameterized policy $\pi_{\boldsymbol{\theta}}$ with GP and generating actions with GP regression gives a direct mapping from states to actions. However, when two robots exhibit the same behavior in the same task, each robot's policy has to be modeled with a different GP and its parameters learned in two independent processes. Thus, the knowledge acquiring during the learning process of one robot cannot be easily exploitable and transferable to another. On the other hand, in the following section we will see how we can learn a latent variable model with GP for constructing an alternative space for the demonstrated behavior. This new model will be used to model a robotic skill that can be exploited for transfer learning.

3.2.2 Learning a latent space with GP-LVM

The Gaussian process latent variable model (GP-LVM) is a non-parametric probabilistic model which performs a non-linear dimensionality reduction over observed data [68]. Let us define $\mathbf{Z} \in \mathbb{R}^{N \times \mathrm{D}_{\mathcal{Z}}}$ as the latent matrix (with dimension $\mathrm{D}_{\mathcal{Z}} < \min(\mathrm{D}_{\mathcal{S}}, \mathrm{D}_{\mathcal{A}})$), $\mathbf{K} \in \mathbb{R}^{N \times N}$ as a kernel, and $\boldsymbol{Y} \in \{\mathbf{S}, \mathbf{A}\}$ to represent either the state or action data. In GP-LVM, compared to other latent variable models, the marginalization is carried out over the model parameters instead of the latent variables, which are optimized. Specifically, the marginal likelihood for $\mathbf{Y}$ given the latent coordinates $\mathbf{Z}$ is specified by:

$$\begin{aligned} p(\mathbf{Y}|\mathbf{Z};\boldsymbol{\theta}_Y) &= \prod_{d=1}^{D_Y} \mathcal{N}(\mathbf{Y}_{:,d}|\mathbf{0},\mathbf{K}_Y) \\ &= \frac{1}{\sqrt{(2\pi)^{ND_Y}|\mathbf{K}_Y|^{D_Y}}} \exp\left(-\frac{1}{2} tr(\mathbf{K}_Y^{-1}\mathbf{Y}\mathbf{Y}^{\mathsf{T}})\right) \end{aligned}$$

where $\mathbf{Y}_{:,d}$ denotes the d-th column of $\mathbf{Y}$, D_Y the dimension of either the state or action spaces, and $\boldsymbol{\theta}_Y$ the hyperparameters of the kernel $\mathbf{K}_Y$. The maximization of this marginal likelihood not only provides us the optimal hyperparameters, but also the optimal set of latent points, that is,

$$\{\mathbf{Z}^*, \boldsymbol{\theta}_Y^*\} = \underset{\mathbf{Z},\boldsymbol{\theta}_Y}{\operatorname{argmax}}\, p(\mathbf{Y}|\mathbf{Z};\boldsymbol{\theta}_Y). \tag{3.5}$$

The demonstrated behavior is composed for both state and action data. While GP-LVM provides us the latent coordinates of either $\mathbf{S}$ or $\mathbf{A}$, it does not account for both of them simultaneously.

To conform with the previous requirement, we make use of a shared GP-LVM, an extended version of the GP-LVM, that assumes that multiple observational spaces [2] share a common latent space [18]. With J observational spaces, the marginal likelihood to maximize is then given by

$$\{\mathbf{Z}^*, \{\boldsymbol{\theta}_{Y^{(j)}}^*\}_{j=1}^J\} = \underset{\mathbf{Z},\{\boldsymbol{\theta}_{Y^{(j)}}\}_{j=1}^J}{\operatorname{argmax}} \; p(\{\mathbf{Y}^{(j)}\}_{j=1}^J|\mathbf{Z};\{\boldsymbol{\theta}_{Y^{(j)}}\}_{j=1}^J), \tag{3.6}$$

where the hyperparameters $\boldsymbol{\theta}_j$ of each kernel, and the latent coordinates $\mathbf{Z}$ are jointly optimized.

[2] In this context, the term observational space refers to the statistical meaning, that is the space of those variables that are directly observed and not inferred (the opposite of latent variables).

In order to preserve local distances and have a smooth mapping from one of the observational spaces to the latent space, back-constraints are introduced. With back-constraints placed on one of the spaces [69], the objective function to maximize becomes

$$\{\boldsymbol{\theta}_Z^*, \{\boldsymbol{\theta}_{Y^{(j)}}^*\}_{j=1}^J\} = \underset{\boldsymbol{\theta}_Z, \{\boldsymbol{\theta}_{Y^{(j)}}\}_{j=1}^J}{\operatorname{argmax}} \; p(\{\mathbf{Y}^{(j)}\}_{j=1}^J | \boldsymbol{\theta}_Z; \{\boldsymbol{\theta}_{Y^{(j)}}\}_{j=1}^J), \tag{3.7}$$

with $\mathbf{Z} = g(\mathbf{Y}^{(j)}; \boldsymbol{\theta}_Z)$ where g is a function from one observational space $\mathbf{Y}^{(j)}$ to the latent space $\mathbf{Z}$ and parameterized by weights $\boldsymbol{\theta}_Z$, which are learned during the optimization process. Placing back-constraints either on $\mathbf{S}$ or $\mathbf{A}$ results in different performances and behaviors which will be further described in section 3.4.2.

Once the model has been trained, given a new data point $\hat{\mathbf{y}}^{(j)}$ in the space $\mathbf{Y}^{(j)}$, predictions in other spaces are realized by first projecting $\hat{\mathbf{y}}^{(j)}$ to the latent space, and then projecting it to the other spaces. For example, assuming that back-constraints are placed on the state space, the state-to-latent mapping is performed using the learned parametric function $g(\hat{\mathbf{s}}; \boldsymbol{\theta}_Z)$, and then the latent-to-action mapping with the shared GP-LVM model. Alternatively, the state-to-latent mapping might also be achieved by carrying out a GP regression once a non-back-constrained shared GP-LVM model has been learned, as performed in [20].

3.3 Shared latent spaces for transfer learning of robot skills

3.3.1 Modeling a robotic skill as a shared latent space

Let us assume that the state-action trajectory $\tau = \{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\}$ generated by a robot following a particular policy π in a time horizon T, can also be expressed as a set of state-action pairs $\tau = \{(\mathbf{s}_1, \mathbf{a}_1), \ldots, (\mathbf{s}_T, \mathbf{a}_T)\}$, where $(\mathbf{s}_t, \mathbf{a}_t)$ is the state-action pair at time-step t. Given a set of D trajectories, $\mathcal{D} = \{\tau_1, \ldots, \tau_D\}$, collected with a policy π for a particular task $\mathcal{T}$, we can learn a shared latent space $\mathcal{Z} \in \mathbb{R}^{D_\mathcal{Z}}$. The space $\mathcal{Z}$ is called *skill space*, and a variable $\mathbf{z}$ in this space is called *skill variable*. A skill variable generates a state-action pair, thus a graphical model of this process is depicted in Figure 3.1a.

The skill variables are obtained by capturing the statistical dependence between the states and actions generated by the policy π. And, as we can see in Figure 3.1a, the model assumes that the joint distribution of state and action is factorized into independent conditional distributions given the skill variable. Thus, this model encodes the behavior captured in the demonstrations and allows to generate new state-action pairs similar to the demonstrations.

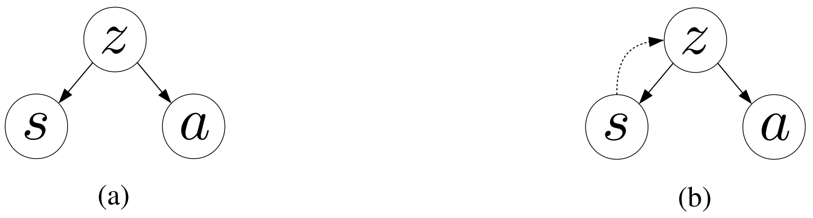


Figure 3.1 *Shared latent variable model representing a skill variable.* (a) shows the generation process of both state **s** and action **a** from a skill variable **z**. (b) includes a dashed line that represents the back-mapping from state **s** to skill variable **z**.

In addition, it is also convenient to learn a back-mapping from states to skill variables (Figure 3.1b) in order to obtain the skill variable that generated a particular state.

As mentioned in section 2.1.1, it is possible to infer the policy π that generated the set of trajectories $\mathcal{D}$, by formulating a regression problem, for example with (3.4), and learning a direct state-to-action mapping (Figure 3.2a). In contrast, with the proposed skill model (Figure 3.2b), a state is transformed to a skill variable which in turn generates the action. In this sense, the policy is indirect because the action is inferred from the skill variable. That is, the policy is reduced to a conditional distribution that uses only the skill variable.

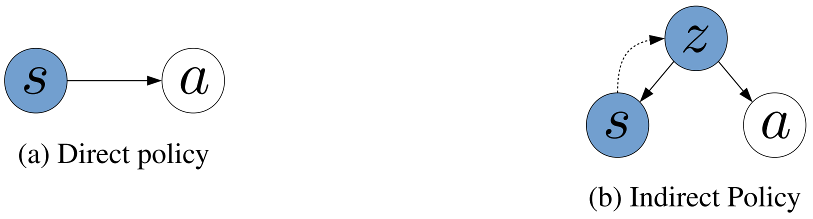


Figure 3.2 *Action generation through a skill variable.* Instead of a direct policy (Fig. a) modeled with the conditional probability $\pi(\mathbf{a}|\mathbf{s})$, an indirect policy $\pi(\mathbf{a}|\mathbf{s},\mathbf{z})$, with the skill variable $\mathbf{z}$, is proposed (Fig. b). This indirect policy is reduced to $\pi(\mathbf{a}|\mathbf{z})$ by conditional independence.

3.3.2 Exploiting robotic skill spaces for transfer learning

When the same skill has to be learned by multiple robots it is necessary to collect a set of trajectories for each robot and then carry out the learning process separately. The state representation of the task $\mathcal{T}$ can even be the same, but a different learning process is required because each robot has specific characteristics that make the action space unique, and, as a consequence, it is required a particular policy for each robot.

However, regardless of the particular actions that each robot takes, the state representation, and the evolution it undertakes during the task execution, is similar across the robots. As a result, one expects that some knowledge is common and therefore the learning process carried out by one robot may be exploited when training the others. This section will describe how a transfer learning process can be done by exploiting the skill space proposed in section 3.3.1.

The skill space is (nearly) invariant among different robots if two assumptions are made. First, the state representation of the environment should be the same among the robots. In other words, the variables considered in the decision process are the same for all the robots: $\mathbf{s}^{(1)} = \mathbf{s}^{(2)} = \cdots = \mathbf{s}^{(J)}$; $\forall j \in \{1,...,J\}$, where j is the index of each robot. Secondly, the influence of an action to the evolution of this state are both independent of the robot executing the task $\mathcal{T}$: $p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t^{(1)}) = p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t^{(2)}) = \cdots = p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t^{(J)})$; $\forall j \in \{1,...,J\}$. As a result, given a horizon T, the state trajectories generated with each robot's policy, $\{\mathbf{s}_1,\mathbf{s}_2,\ldots,\mathbf{s}_T\}$, are also the same across the robots.

Therefore, considering the previous assumptions, it is possible to formulate the problem of learning the same skill for task $\mathcal{T}$ among different robots as a transfer learning one, where the state-latent map, therefore also the skill space, of one robot (source domain) can be used to learn the latent representation for others (target domains). In other words, teaching a skill with BC can be carried out by first learning the skill space with the demonstrations of one robot, and then exploiting it for training other ones. By transferring this prior latent knowledge from one robot to another, we expect to achieve a faster learning rate along with good imitation performance. This transfer learning process is depicted in Figure 3.3.

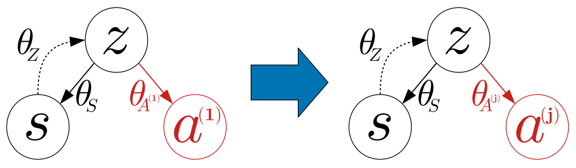


Figure 3.3 *Transfering the skill space of a robot.* When the same skill for task $\mathcal{T}$ has to be learned by different robots, the parameters $\boldsymbol{\theta}_S$ and $\boldsymbol{\theta}_Z$ for a robot i are fixed and only the parameters $\boldsymbol{\theta}_A^{(j)}$ trained for a new robot j, $\forall j \neq i$.

3.3.3 Transfer learning of robotic skills with shared GP-LVMs

In this section, first the the skill space is modeled with the shared GP-LVM described in section 3.2.2. Moreover, a model which is first fully trained on a specific robot, is transferred to other robots with partial re-training, as shown in Figure 3.4.

First, let us denote N as the number of data points in our demonstration training set, $\mathbf{S} = [\mathbf{s}_1 \cdots \mathbf{s}_N]^\mathsf{T} \in \mathbb{R}^{N\times \mathrm{D}_{\mathcal{S}}}$ as the state data matrix, and $\mathbf{A}^{(j)} = [\mathbf{a}_1^{(j)} \cdots \mathbf{a}_N^{(j)}]^\mathsf{T} \in \mathbb{R}^{N\times \mathrm{D}_{\mathcal{A}^{(j)}}}$; $\forall j \in \{1,...,J\}$ as the j-th robot action data matrix where J denotes the total number of robots that are learning the same skill. Assuming back-constraints on the state, in order to preserve

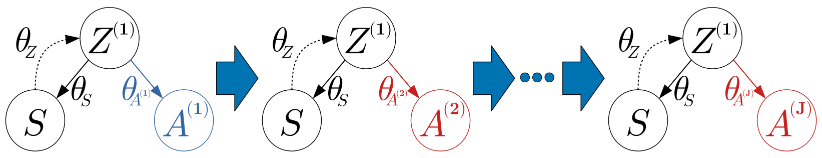


Figure 3.4 *Transfering a learned shared GP-LVM to other robots.* First, a shared GP-LVM fully trained on the action data of one robot $\mathbf{A}^{(1)}$ jointly optimizes the hyperparameters $\boldsymbol{\theta}_S$ and $\boldsymbol{\theta}_{A^{(1)}}$, and either the latent variables $\mathbf{Z}^{(1)}$ or the back-constraints parameters $\boldsymbol{\theta}_Z$. Later, this model is reused with the action data of a second robot $\mathbf{A}^{(2)}$. The transfer learning process is carried out by keeping fixed the hyperparameters $\boldsymbol{\theta}_S$ and the latent coordinates (or the parameters $\boldsymbol{\theta}_Z$ in the case of back-constraints) and only optimizing the hyperparameters $\boldsymbol{\theta}_{A^{(2)}}$ for the new latent-to-action mapping. This process is repeated for any other robot that learns the same skill.

local distances, the hyperparameters of the shared GP-LVM model on one particular robot, and the parameters of the state-to-latent mapping, are jointly optimized by maximizing the marginal likelihood:

$$\{\boldsymbol{\theta}_Z^*, \boldsymbol{\theta}_S^*, \boldsymbol{\theta}_{A^{(1)}}^*\} = \underset{\boldsymbol{\theta}_Z, \boldsymbol{\theta}_S, \boldsymbol{\theta}_{A^{(1)}}}{\operatorname{argmax}} \; p(\mathbf{S}, \mathbf{A}^{(1)} | \boldsymbol{\theta}_Z; \boldsymbol{\theta}_S, \boldsymbol{\theta}_{A^{(1)}}). \tag{3.8}$$

The parameters $\boldsymbol{\theta}_Z$ allow us to obtain the optimal latent variables for the skill by $\mathbf{Z}^* = h(\mathbf{S}; \boldsymbol{\theta}_Z^*)$. This pretrained model is then transferred to another robot by fixing the latent variables and the state-to-latent mapping, and optimizing only the latent-to-action mapping. This process is carried out for any other robot $j \in \{2, ..., J\}$, by optimizing:

$$\boldsymbol{\theta}_{A^{(j)}}^* = \underset{\boldsymbol{\theta}_{A^{(j)}}}{\operatorname{argmax}} \, p(\mathbf{S}, \mathbf{A}^{(j)} | \mathbf{Z}^*; \boldsymbol{\theta}_S^*, \boldsymbol{\theta}_{A^{(j)}}). \tag{3.9}$$

As only the latent-to-action mapping is optimized, this naturally leads to a faster learning process for the new robots. When all the hyperparameters $\{\boldsymbol{\theta}^*{}_{A^{(i)}}\}_{i=1}^J$ are obtained, it is possible to build a shared GP-LVM in which all the other robots are appended to the initial fully trained model (see Figure 3.5).

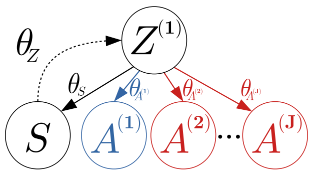


Figure 3.5 *Shared GP-LVM among many robots learned with transfer learning.* The model is built by following the process depicted in Figure 3.4. After training, given a new state the model can generate the actions for all the robots.

In order to obtain a sucessful transfer, the back-constraint imposed on the state was motivated by four reasons. Firstly, the state-to-action mapping has to be smooth. While a GP-LVM already provides a smooth mapping from the latent space to the observational spaces, the converse is not true [69]. Thus, by ensuring a smooth mapping from the state to the skill space with back-constraints, a final smooth state-latent-action is obtained. Secondly, constraining the skill space (and thus the skill parameters) by any means would be beneficial as it would (at the expense of an introduced bias) decrease the variance and thus make it more robust to overfitting. Thirdly, when transfering the model to another robot, the state data is fixed while the action data changes. Imposing back-constraints on the state makes the model less dependent on the particular robot actions, and thus more stable and robust. Finally, it is exploited the ability to jointly optimize the input-to-latent mapping along with the model. The skill variables are hence indirectly optimized while being still dependent on the back-constraint function. This is relevant as this mapping will be used later when predicting the skill coordinates given new state data.

Exploiting the latent coordinates to transfer acquired knowledge to other robots is a promising approach. It has the potential to lead to faster learning and similar reproduction performance compared to their fully trained counter parts, as confirmed in the experiments conducted in section 3.4.2.

Multi-robot systems with shared GP-LVMs

The previous approach allows transfer learning of the robot skill, but it can also be exploited for other kind of models. A multi-robot model can be built if we consider that the state space and all robot's action spaces are generated by a common latent space (see Figure 3.6). The optimization process has to make a trade-off between the latent coordinates and the hyperparameters of each kernel for the state and all the robot actions. Thus, the solution is defined by

$$\{\boldsymbol{\theta}_Z^*, \boldsymbol{\theta}_S^*, \{\boldsymbol{\theta}_{A^{(i)}}^*\}_1^J\} = \underset{\boldsymbol{\theta}_Z, \boldsymbol{\theta}_S, \{\boldsymbol{\theta}_{A^{(i)}}\}_1^J}{\operatorname{argmax}} \; p(\mathbf{S}, \{\mathbf{A}^{(i)}\}_1^J | \boldsymbol{\theta}_Z; \boldsymbol{\theta}_S, \{\boldsymbol{\theta}_{A^{(i)}}\}_1^J). \tag{3.10}$$

In contrast to the model presented in section 3.3.3, the latent coordinates and *all* the hyperparameters of this multi-robot model should be jointly optimized, therefore no prior information from previous robots may be exploited. As a result, if a new robot requires to learn the same skill, the problem in (3.10) should include new hyperparameters $\boldsymbol{\theta}_{A^{(i)}}$ for this new robot and the new model trained with the new robot's action data $A^{(i)}$.

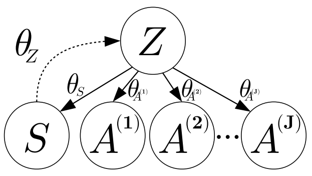


Figure 3.6 *Multi-robot learning model.* In this model the latent coordinates $\mathbf{Z}$, the hyperparameters $\boldsymbol{\theta}_S$ and the hyperparameters of each robot actions, $\{\boldsymbol{\theta}_{A^{(i)}}\}_1^J$, are jointly optimized. The model is less biased to a specific initial selected robot because it compromises among all the robots. However, if a new robot requires to learn the same skill, the whole model should be retrained.

3.4 Experiments

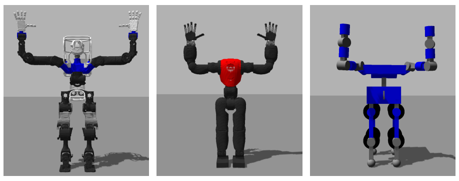
This sections details the experiments conducted in order to validate the formulation proposed in the previous section. The description of the experiments is presented in section 3.4.1 and the obtained results are reported and analyzed in section 3.4.2.

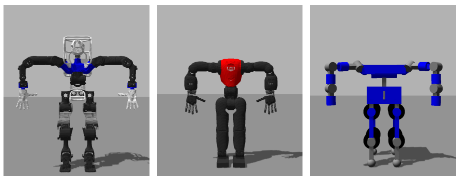
3.4.1 Setup description

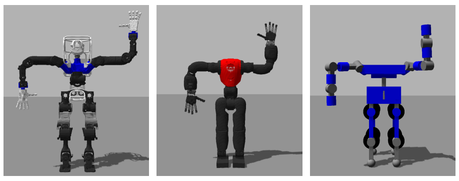
The proposed framework was evaluated in a teleoperation task, where three simulated robots WALK-MAN [70], COMAN [71] and CENTAURO [72] were required to reproduce bimanual movements of a human wearing a Xsens MVN BIOMECH suit. The state, with $\mathrm{D}_{\mathcal{S}} = 18$, is represented by the arms configuration (3 DoF for each shoulder, upper arm, and forearm), and sensed by the motion capture system. On the other hand, all the three simulated robots have 7 position-controlled revolute joints in each arm, thus $\mathrm{D}_{\mathcal{A}^{(1)}} = \mathrm{D}_{\mathcal{A}^{(2)}} = \mathrm{D}_{\mathcal{A}^{(3)}} = 14$.

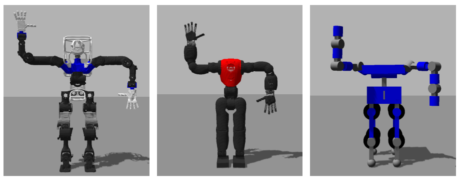
Because all the three robots are simulated, most of the typical imitation learning interfaces were not valid for collecting the required demonstrations data. However, by exploiting the assumptions that the state evolution in the demonstrations is the same, and independent of the robot action space, both state and action data were collected independently.

The acquisition of the action data began by defining a set of 16 key bimanual poses for each robot, as depicted in Figure 3.7. Then, robot trajectories between these poses were generated using a quintic Hermite interpolator with zero initial and final velocities and accelerations. The robots movements were simultaneously executed in the simulation, and the human mimicked the robots' motion. To collect variability in the data, the human went back and forth between different poses three times. During this process, the joint values of the human and the desired joint commands of the robot were recorded at 40 Hz. The joint trajectories collected for each robot represented their respective action data of the task, and the data captured by the motion capture suit, the state data.


(a) Robots' Pose 1


(b) Robots' Pose 2


(c) Robots' Pose 3


(d) Robots' Pose 4

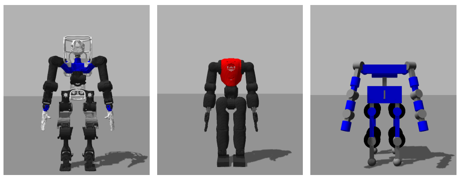

(e) Robots' Neutral Pose

Figure 3.7 Five of the sixteen robots' poses defined for each robot. From left to right: WALK-MAN, COMAN and CENTAURO.

Once the demonstrations data was collected, all the trajectories starting from the neutral pose were grouped as training set, and the trajectories between different poses excluding the neutral pose as test set. Finally, the trajectories categorized as training data were subsampled with the objective of reducing the number of data points.

The collected dataset was used in the experiments to (i) corroborate the hypothesis that the skill spaces can be exploited to transfer prior information, (ii) analyze the benefits of applying back-constraints in a shared GP-LVM between state and action, (iii) compare the multi-robot learning models described in section 3.3.3 with independent shared GP-LVMs for each robot. These models were implemented, trained and tested by extending the GPy framework [73].

Three shared GP-LVMs were modeled to analyze the benefits of back-constraints. The first model did not have back-constraints and the second model presented back-constraints from the robot joints, $\mathbf{A}^{(j)}$, to the skill space, $\mathbf{Z}$. In both, the mapping from human data $\mathbf{S}$ to $\mathbf{Z}$ was carried out with a Gaussian process regression, once the shared model was learned. The third model had back-constraints from $\mathbf{S}$ to $\mathbf{Z}$ so the corresponding mapping was performed by the learned parametric function $g(\hat{\mathbf{S}}; \boldsymbol{\theta}_Z)$, represented by a SE kernel-based mapping.

The latent locations in the shared GP-LVM without back-constraints were initialized by averaging the principal component analysis (PCA) solutions of $\mathbf{S}$ and $\mathbf{A}^{(j)}$. This initialization was not required in the models using back-constraints where the latent locations are parameterized by $\boldsymbol{\theta}_Z$.

The shared GP-LVM shown in Figure 3.5 was modeled by replacing the corresponding joint angles of one robot by another, and training only the hyperparameters of the latent-to-action mapping. On the other hand, the multi-robot model depicted in Figure 3.6 was characterized by a multi-output structure containing all the joint angles of the three robots.

In all the shared GP-LVMs, the dimension of the shared latent space was set to $D_{\mathcal{Z}} = 5$, a value that reduced the dimension of the skill space while still achieving a performance similar to a GP regression representing a direct state-action mapping. The SE kernel function with ARD was used, and the hyperparameters along with the latent positions (in the case without back-constraints) were optimized with the L-BFGS-B algorithm.

3.4.2 Results

To corroborate the hypothesis that the skill space of the demonstration can be exploited to transfer prior knowledge, the proposed model (described in section 3.3.3 and depicted in Figure 3.4) was evaluated first without back-constraints. The results for each robot are reported in Figure 3.8. The plots show the reconstruction errors on a test dataset corresponding to the proposed shared GP-LVM for transfer learning, a model fully trained on a single robot dataset (used as baseline), and the alternative multi-robot model shown in Figure 3.6. Both the baseline and multi-robot models were averaged over 10 runs while the proposed model was averaged over 10 runs for each run of the fully trained model, thus a total of 100 runs.

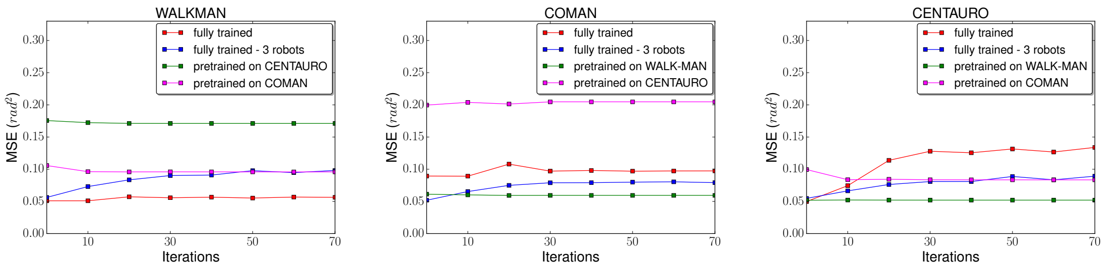


Figure 3.8 Mean squared error of the pretrained and fully trained models with no back-constraints for the three robots on the test dataset.

Figure 3.6 shows that the mean squared error (MSE) achieved by the baseline model on one robot is similar to the performance of the proposed model pretrained on the same robot. This observation is consistent with our expectation that the optimization of the latent-to-action mapping of our pretrained models should not influence excessively the results, as the latent coordinates and input-to-latent mapping remain unchanged. While we indeed observe faster convergence for the proposed shared GP-LVM, because of the smaller amount of hyperparameters to optimize, a similar test error is not observed between the different models. To address this issue, back-constraints were incorporated into the model.

The results obtained by using models with back-constraints for each robot are reported in Figure 3.9. In contrast to the results shown in Figure 3.8, the obtained results are significantly better, and are similar across the different models. Moreover, the pretrained models still exhibit faster convergence compared to the fully trained ones.

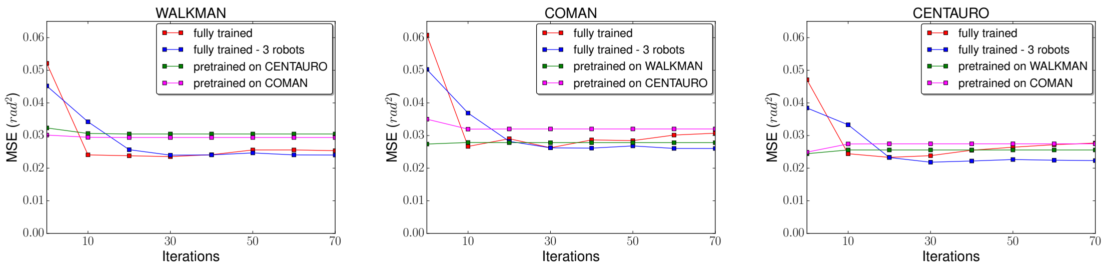


Figure 3.9 Mean squared error of the pretrained and fully trained models with back-constraints set on the input for the three robots on the test dataset.

To further confirm the benefits of using back-constraints, an additional test comparing the fully trained model on each robot with back-constraints on the input, output, and without back-constraints, is performed. The results depicted in Figure 3.10 validate the assumption that back-constraints in general are beneficial compared to the non-back-constraints case. More importantly, they show that imposing back-constraints on the state space results in the best performance.

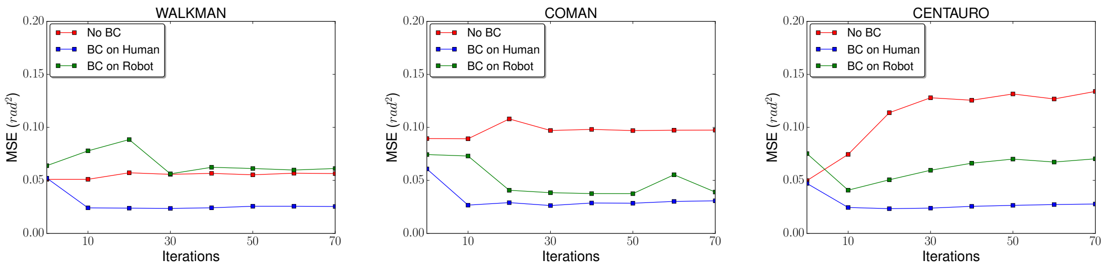


Figure 3.10 Mean squared error showing the performance without back-constraints (red), with back-constraints placed on the input (blue) or on the output (green) for the three robots on the test dataset.

Having confirmed that placing back-constraints on the state space is beneficial for the transfer to be successful, it is analyzed a particular case where the model trained on WALKMAN is then transferred to COMAN. The objective is to analyze the structure of the latent space and the predictions made by the proposed model given new human input data. The learned shared latent space for the WALK-MAN robot with back-constraints on the input is depicted in Figure 3.11. The represented poses are the same as those shown in Figure 3.7. The lines represent the resulting latent trajectories for some of the human test data. While these trajectories show clearly the forth and back movement, they are not smooth because of the noisy human input data.

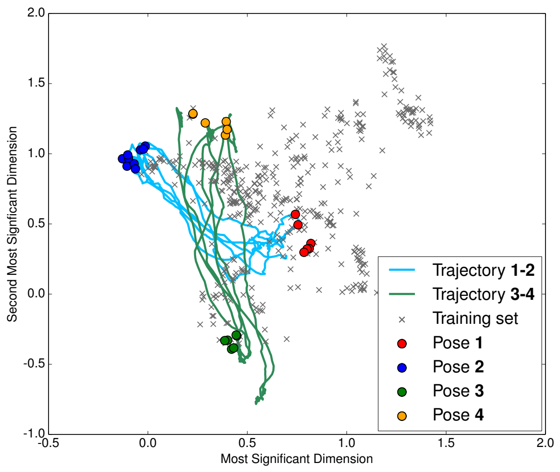


Figure 3.11 Trajectories in the learned skill space of a human and the WALK-MAN robot with back-constraints on the state space. The circles represent four of the predefined key poses, while the crosses represent the skill positions learned with the training data movements.

The predictions made by the model given the trajectories of the human arms when moving back and forth between the pose 1 and 2 are reported in Figure 3.12. This corresponds to the blue trajectory in the skill space shown in Figure 3.11. The pose 1 is depicted in Figure 3.7a while the pose 2 is similar to the former one but with the shoulders rotated 180° in the sagittal plane. The model was first fully trained on WALK-MAN, then transferred to COMAN where the latent-to-output mapping was learned. It can be seen that the predicted robot joint trajectories follow the desired pattern, are robust to a certain extent to noisy input data, and can deal satisfactorily with the difference between the robot kinematic structures.

3.5 Challenges

It has been experimentally shown that modeling a skill space with a shared GP-LVM for transfer learning is a promising approach. With the method described in this chapter, transferring part of the model learned with one robot offers faster convergence rates for

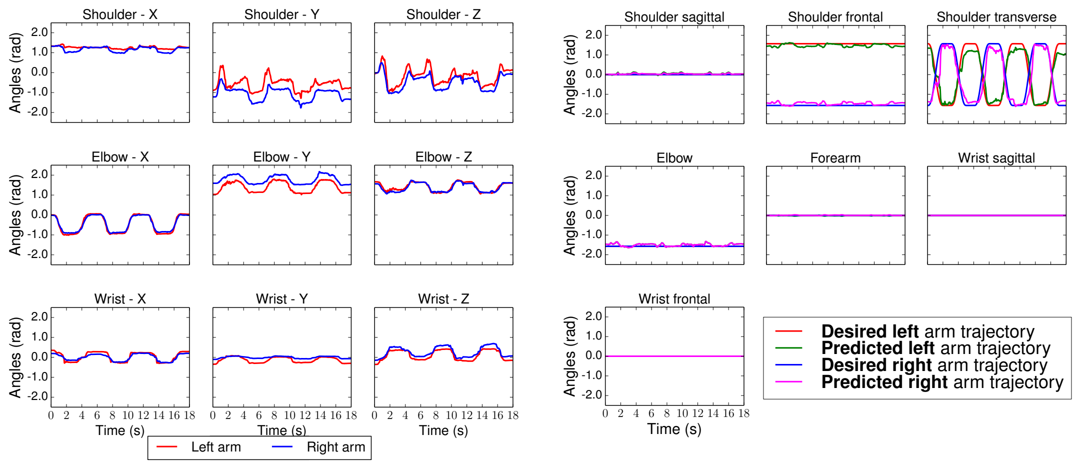


Figure 3.12 *COMAN joint trajectories generared for COMAN.* Human left and right arm trajectories (left), and corresponding predicted left and right robot arm trajectories (right). A shared GP-LVM is trained on WALK-MAN with back-constraints set on the input, then transferred and partially trained on COMAN. The robot trajectories are the movements performed by COMAN which result from this transfer learning. The human trajectories are part of the test dataset, and the corresponding latent trajectories are depicted in blue in Figure 3.11.

new models, without detriment of the final performance of the new policies. However, some challenges need to be addressed to further exploit the potential of this work.

The first open issue concerns the latent space dimensionality, which is currently set by the user. In the experiments, the latent dimension size is manually set motivated by the desire to reduce as much as possible the dimensionality of the latent space while still achieving good performances close to a GP regression. Choosing a specific dimensionality can be cumbersome as it requires testing different latent dimensions to satisfy different criteria. One possibility to overcome this limitation is to use variational Bayesian techniques to automatically estimate the dimensionality of the skill space.

The experiments were performed with robots with similar kinematic structure, and with equivalent action spaces, robots with significant differences in their actions should be further investigated. However, in view of the obtained results, if the skill space is properly built the proposed method should still lead to good results.

A strong assumption of the proposed approach is that the state representation and the effect of different robot actions on it are the same. There are several tasks where the state should include data that is robot-specific, as a result, the variables in the skill space of one robot should also generate the states of a new robot. However, this shared latent space may not be enough for explaining the differences among the robot-specific states. A possible solution is to exploit more complex latent models, such as the Non-Consolidating-

Component-Analysis (NCCA) model [18]. With this model the common evolution among the robot trajectories may be explained by the skill space, and the robot-specific behavior by robot-specific latent spaces.

3.6 Summary

In this chapter we have seen how instead of exploiting demonstration trajectories for learning directly a rational robotic policy, it is possible to assume that this particular skill can be represented by a shared latent variable model, that explains both state and action data from the demonstrations. A variable in the shared latent space of the model is the only one that generates the action. Thus, by learning a mapping state-latent, it is possible to obtain a state-latent-action policy. When the state representation of the environment and the influence of an action to the evolution of this state are both independent of the robot executing the task, the latent representation of one robot is used to learn the latent representation for others.

A shared GP-LVM was proposed for learning this latent space, so-called skill space, and transferred from one robot to another. It was shown the importance of representing each skill variable as a smooth mapping (back-constraint) from the state, in order to obtain a successful transfer. The experiments were conducted in a teleoperation task, where three different robots have all to mimic human movements. The results shown that the transfer learning process allows faster convergence rates without detriment of the performance of the robot-specific policies.

Chapter 4

CONCURRENT DISCOVERY OF COMPOUND AND COMPOSABLE POLICIES

Reinforcement learning (RL) is a general framework that allows an agent to autonomously discover rational, that is optimal, behaviors by interacting with its environment. However, when applied to robotics scenarios some specific challenges arise, such as high-dimensional continuous state-action spaces, real-time requirements, delays and noise in sensing and execution, and expensive (real-world) samples [5]. Recently, several authors have overcome some of these challenges by using deep neural networks (NN) as parameterized policies for generating rich behaviors in end-to-end frameworks [74, 75]. Nevertheless, learning the high number of parameters of deep NN in complex tasks involves a large number of interactions with the environment that compromises the real-world sample challenge.

Several algorithms have been proposed to improve sample efficiency of model-free deep RL by making a better use of the sample information (data-efficiency), obtaining more information from data (sample choice) and improving several times the policy with the same samples (sample reuse) [17]. However, learning a complex robotic task may still be slow or even infeasible when the robot learns from scratch. An appealing approach to deal with this problem is to decompose the task into subtasks that are both simpler to learn and reusable for new problems.

Many tasks in robotics may intuitively be divided into individual tasks, for example, the task of moving an object to a specific location may be decomposed into reaching for the object, grasping it, moving it to the target, and releasing it. Different rational policies can be defined in these *composable* tasks, each one optimizing its criterion for selecting the control actions. Therefore, when a robot is provided with a collection of these composable policies, a new RL problem can be stated as learning how to combine them such that the performance criterion of the complex task is optimized. Note that shifting the complexity of this task learning problem to layers of simpler functionality is mainly studied in the field of hierarchical RL. However, the assumptions of common hierarchical RL algorithms limit their application in several robotic tasks.

Firstly, several hierarchical RL methods decompose the complex RL problem temporally, meaning that during a certain period of time the behavior of the robot is delegated to a specific subtask policy [76]. This temporal decomposition is suitable for robotic tasks that can be divided sequentially, however, many others require the robot to perform individual tasks concurrently (e.g. a manipulator carrying an object and avoiding an obstacle simultaneously). Secondly, common learning methods optimize the individual policies and the compound policy through independent single-task RL processes [77, 78, 79]. Therefore, the robot has to continuously interact with the environment to learn, possibly from scratch, first a collection of composable policies, and only after that, their composition, compromising sample efficiency.

This chapter presents a two-level hierarchical RL approach alternative to a single-task RL process for solving the complex task. First, a set of Gaussian policies, constituting the low level of the hierarchy, are composed at the high level by means of state-dependent activation vectors defined for each policy. These activation vectors allow to consider concurrently actions sampled from all the low-level policies and preferences among specific components. Furthermore, it is proposed in section 4.3 two alternatives for obtaining a compound Gaussian policy as a function of the parameters of the low-level policies and their corresponding activation vectors. Therefore, the behavior in the complex task is not generated directly by actions sampled from a single policy trained in the task. Instead, the behavior is generated by policies specialized in different aspects of the task.

Additionally, instead of learning the different components of the model in independent processes, the algorithm detailed in section 4.4 exploits the off-policy data generated from the compound policy for learning the high-level policy and indirectly the low-level policies within the same process. This multi-task formulation is built on a maximum entropy RL approach to favor exploration during the learning process. The results obtained from the conducted experiments suggest that the experience collected with the compound policy permits not only to solve the complex task but also to obtain useful composable policies that successfully perform in their respective tasks.

4.1 Related work

Complex problems in RL usually involve either tasks that can be hierarchically organized into subtasks, scenarios requiring concurrent execution of several tasks, and tasks with large or continuous state-action spaces [80]. Hierarchical RL approaches split a complex task into a set of simpler elementary tasks [81]. Some of these methods have been successfully applied in robotics by exploiting temporal abstraction, where the decision to invoke a particular task is not required at each time step but over different time periods [76, 82]. As a consequence, these methods assume that a high-level policy, which selects the subtask, and low-level policies, which select the action, are executed in different time scales. In contrast, the approach proposed in this chapter considers that the decisions at both levels of the hierarchy are executed at each time step.

The temporal abstraction assumption in most hierarchical RL methods also involves that during a certain period of time the robot only performs a particular task. RL problems requiring policies that solve several tasks at the same time are commonly stated as multiobjective or modular RL problems [83, 84]. The policies of all these subtasks may be combined using weights describing the predictability of the environmental dynamics [85], or the values obtained from the desirability function in a linearly-solvable control context [86]. Another alternative is to combine action-value functions of composable tasks, and then extract a policy from this combined function [79]. The latter paper is similar to this chapter's approach since the composable policies are also optimizing an entropy-augmented RL objective, however, their combination is carried out at the level of action-value functions unlike the proposed policy-based approach. Moreover, their composable policies have been previously learned in independent processes, in comparison to the algorithm proposed here where both compound and composable policies are improved in the same RL process.

Exploiting the experience collected using a particular policy (so-called behavior policy) to concurrently improve several policies is a promising strategy that has been previously explored in literature. The Horde architecture shows how independent RL agents with same state-action spaces can be formulated for solving different problems [87]. This approach proposed by Sutton et al. is relevant as it exploits its off-policy formulation for improving all the policies in parallel.

Several works extended the Horde formulation and used deep NN for dealing with high dimensional or continuous state-action spaces, and also to provide a modularity that can be exploited for modeling and training independent RL problems [88, 89, 90, 91]. The Intentional-Unintentional agent [88], for example, shows how deep NN policies and value functions can be learned even with an arbitrarily selected behavior policy. However, this policy can also be chosen at each time step by following a hierarchical objective, where the selection process can be improved as a function of the performance in the complex task [91]. Note that the policies in Cabi et al. [88] and Yang et al. [91] are deterministic, and then the exploration should be carried out by adding a noise generated from an Ornstein-Uhlenbeck process. In contrast, all the policies in this chapter are stochastic and the exploration is generated directly from the behavior policy, that is the stochastic high-level policy.

Most of the notation and the approach proposed in this chapter are inspired by the Scheduled Auxiliary Control (SAC-X) method [92]. SAC-X solves complex tasks based on a collection of simpler individual tasks, and learns from scratch, both high- and low-level policies simultaneously. However, this method considers temporal abstraction in the hierarchy, and therefore the high-level policy is a scheduler that occasionally selects one low-level policy. Therefore, the policies at the low level of the hierarchy can only be executed sequentially and run at time scales different from that of the high-level policy. This methodology differs from this chapter's framework that executes low- and high-policies concurrently.

4.2 Preliminaries

Recall from chapter 2 that the sequential decision making process of a robot could be modeled by a Markov decision process (MDP) $\mathcal{M}$, defined by the tuple $(\mathcal{S}, \mathcal{A}, p_s, r)$, where $\mathcal{S} \subset \mathbb{R}^{\mathrm{D}_\mathcal{S}}$ and $\mathcal{A} \subset \mathbb{R}^{\mathrm{D}_\mathcal{A}}$ are continuous state and action spaces of dimensionality $\mathrm{D}_\mathcal{S}$ and $\mathrm{D}_\mathcal{A}$, respectively. At each time step t, the robot selects an action $\mathbf{a}_t \in \mathcal{A}$ according to a policy π which is a function of the current state of the environment $\mathbf{s}_t \in \mathcal{S}$. After this interaction, the state of the environment changes to $\mathbf{s}_{t+1} \in \mathcal{S}$ with a probability density $p_s = p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and the robot receives a reward according to the function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The robot's goal is to maximize the expected infinite-horizon discounted return $G(\tau)$ of the trajectory $\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots\}$, induced by its policy π. That is to say $J(\pi) = \mathbb{E}_\tau [G(\tau)] = \mathbb{E}_\tau [\sum_{t=0}^{\infty} \gamma^t \, r(\mathbf{s}_t, \mathbf{a}_t)]$.

4.2.1 Maximum entropy reinforcement learning

The exploration required for a robot to generate behaviors that produce high return can be directly obtained by the stochastic policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$, where the randomness of the actions $\mathbf{a}_t$ given the state $\mathbf{s}_t$ can be quantified by the entropy of the policy. *Maximum entropy reinforcement learning* or entropy regularized RL [93, 94] is a formulation that augments the previous RL objective by including the entropy of the robot's policy

$$J(\pi) = \mathbb{E}_{\tau}\left[\sum_{t=0}^{\infty} \gamma^t \left(r(\mathbf{s}_t, \mathbf{a}_t) + \alpha\mathcal{H}(\pi(\cdot|\mathbf{s}_t))\right)\right],$$

where $\mathcal{H}(\pi(\cdot|\mathbf{s}_t))$ denotes the entropy of an action $\mathbf{a}_t$ with distribution $\pi(\mathbf{a}_t|\mathbf{s}_t)$ and is computed as $\mathcal{H}(\pi(\cdot|\mathbf{s}_t)) = \mathbb{E}_{\mathbf{a}_t \sim \pi}[-\log \pi(\cdot|\mathbf{s}_t)]$. The parameter α controls the stochasticity of the optimal policy. Note that the conventional RL objective is recovered in the limit $\alpha \to 0$.

4.2.2 Soft Actor-Critic algorithm

Soft actor-critic (SAC) is an off-policy actor-critic deep RL algorithm that optimizes stochastic policies defined in the maximum entropy framework [95, 96]. The algorithm is built on a policy iteration formulation that alternates between policy evaluation and improvement steps. In the former, a parameterized soft Q-function $Q_{\boldsymbol{\phi}}$ is updated to match the value of the parameterized policy $\pi_{\boldsymbol{\theta}}$ according to the maximum entropy objective, while in the latter the policy $\pi_{\boldsymbol{\theta}}$ is updated towards the exponential of the updated $Q_{\boldsymbol{\phi}}$. Thus, the soft Q-function parameters $\boldsymbol{\phi}$ can be trained to minimize the soft Bellman residual

$$J_Q(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}}\left[\frac{1}{2}\left(Q_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) - \left(r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1} \sim p_s}[V_{\bar{\boldsymbol{\phi}}}(\mathbf{s}_{t+1})]\right)\right)^2\right]$$

where the value function $V_{\bar{\boldsymbol{\phi}}}$ is implicitly parameterized through a target soft Q-function via

$$V_{\bar{\boldsymbol{\phi}}}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}}[Q_{\bar{\boldsymbol{\phi}}}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t))],$$

and whose parameters $\bar{\boldsymbol{\phi}}$ are obtained as an exponentially moving average (Polyak average) of the soft Q-function parameters $\boldsymbol{\phi}$.

Finally, in the policy improvement step, the policy parameters $\boldsymbol{\theta}$ are updated by maximizing the maximum entropy objective

$$J_{\pi}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}}\left[\mathbb{E}_{\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}}\left[Q_{\boldsymbol{\phi}}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t))\right]\right].$$

4.3 Composition of modular Gaussian policies

Acquiring autonomously a robotic skill for a specific task can be achieved by directly optimizing the maximum entropy RL objective with the experience collected from the task execution. Nevertheless, when a new task defined in the same state and action spaces has to be learned, the robot should interact again with the environment to obtain useful data for improving a new policy for this new task. As a consequence, learning sequentially single-task problems may require an excessive number of interactions in order to learn the parameters of the corresponding policies. Sample efficiency is a key concern in robotics, therefore in this section it is proposed a two-level hierarchical model that seeks to construct a set of simpler and reusable policies in the low level of the hierarchy, and a high-level policy that combines them for solving a more complex task. Thus, the behavior in the complex task is not generated directly by a single policy, instead, it is the result of a composition of several policies specialized in different aspects of the task. Moreover, section 4.4 introduces an algorithm for exploiting better the interaction data and learning both low- and high-level policies all together. This multi-task formulation uses the off-policy generated by the high-level policy in order to learn concurrently the low-level policies via their corresponding maximum entropy objectives.

4.3.1 Hierarchical model for composing modular policies

First, let us assume that several complex tasks can be decomposed into a set of K *composable tasks* $\mathcal{T} = \{\mathcal{T}^{[k]}\}_1^K$. All of them have the same state space, action space and transition dynamics, however, each one is characterized by a specific reward function $r^{[k]}(\mathbf{s}_t, \mathbf{a}_t)$. Thus, the corresponding MDP for each composable task $\mathcal{T}^{[k]}$ is $(\mathcal{S}, \mathcal{A}, p_s, r^{[k]})$. Second, let us assume that stochastic policies defined in these MDPs, called *composable policies*, are conditional Gaussian distributions $\pi^{[k]}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{m}^{[k]}, \mathbf{C}^{[k]})$, with mean vector $\mathbf{m}^{[k]} \in \mathcal{A}$ and diagonal covariance matrix $\mathbf{C}^{[k]} = \mathrm{diag}[\sigma_1^2, \sigma_2^2, \ldots, \sigma_{\mathrm{D}_\mathcal{A}}^2]$.

Let us also define a (possibly more complex) *compound task* $\mathcal{M}$, described by the combination of the tasks in $\mathcal{T}$. As with the composable tasks, $\mathcal{M}$ also shares the same state space, action space and transition dynamics, but it is characterized by the reward $r^{\mathcal{M}}(\mathbf{s}_t, \mathbf{a}_t)$. Finally, a stochastic policy defined in the corresponding MDP $(\mathcal{S}, \mathcal{A}, p_s, r^{\mathcal{M}})$ is named *compound policy* $\pi^{\mathcal{M}}$. This policy reuses the set of composable policies $\Pi = \{\pi^{[k]}\}_1^K$ defined in $\mathcal{T}$, by using the set $\mathcal{W} = \{\mathbf{w}^{[k]}\}_1^K$ where $\mathbf{w}^{[k]} \in \mathbb{R}^{\mathrm{D}_\mathcal{A}}$ is an *activation vector* whose components are used to combine each DoF of the action vector. Therefore, a stochastic compound policy $\pi^{\mathcal{M}}(\mathbf{a}|\mathbf{s})$ is modeled as a two-level hierarchical policy. The generation process of an action

a involves first obtaining the actions from Π in the low level of the hierarchy, and then combining them at the high level of the hierarchy to obtain $\pi^{\mathcal{M}}(\mathbf{a}|\mathbf{s}) = \mathtt{f}(\mathcal{W}, \Pi)$.

There exist several ways to formulate `f`, we here exploit the assumptions made for the composable policies and propose two alternatives for obtaining a policy $\pi^{\mathcal{M}}(\mathbf{a}|\mathbf{s})$ that is also conditional Gaussian and defined in terms of the means $\mathbf{m}^{[k]}$ and covariances matrices $\mathbf{C}^{[k]}$ of the composable policies. The first option is to consider that the action of the compound policy is the convex combination of elements of actions sampled from the composable policies. As the actions for each composable policy are conditionally independent given the states, and each $\pi^{[k]}(\mathbf{a}|\mathbf{s})$ is conditional Gaussian, the resulting action is also normally distributed, and therefore the compound policy $\pi^{\mathcal{M}}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{m}, \mathrm{diag}(\mathbf{c}))$, where the components in $\mathbf{m}$ and $\mathbf{c}$ are computed as

$$\begin{aligned} c_i &= \sum_{k=1}^{K} \left(w_i^{[k]} \, \sigma_i^{[k]} \right)^2, \\ m_i &= \sum_{k=1}^{K} w_i^{[k]} \, m_i^{[k]}, \end{aligned} \tag{4.1}$$

for all $1 \leq i \leq \mathrm{D}_{\mathcal{A}}$, with $w_i^{[k]}$, $m_i^{[k]}$, $\sigma_i^{[k]}$ as the corresponding elements of the activation vector, mean vector, and standard deviation vector for the composable policy $\pi^{[k]}$.

The second alternative for modeling `f` is to consider that each component i in the resulting action vector is obtained from a product of conditional Gaussians

$$\pi^{\mathcal{M}}(a_i|\mathbf{s}) \propto \prod_{k=1}^{K} (\pi^{[k]}(a_i|\mathbf{s}))^{w_i^{[k]}}.$$

As a result, the compound policy is also $\pi^{\mathcal{M}}(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mathbf{m}, \mathrm{diag}(\mathbf{c}))$ where the components in $\mathbf{m}$ and $\mathbf{c}$ are computed as

$$\begin{aligned} c_i &= \left(\sum_{k=1}^{K} w_i^{[k]} / (\sigma_i^{[k]})^2 \right)^{-1}, \\ m_i &= c_i \left(\sum_{k=1}^{K} \left(w_i^{[k]} / (\sigma_i^{[k]})^2 \right) m_i^{[k]} \right), \end{aligned} \tag{4.2}$$

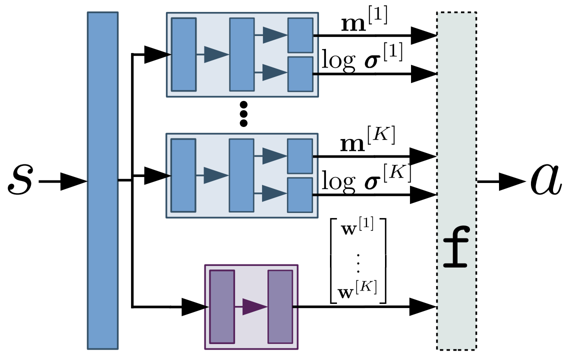
for all $1 \leq i \leq \mathrm{D}_{\mathcal{A}}$, with $w_i^{[k]}$, $m_i^{[k]}$, $\sigma_i^{[k]}$ defined as in the first case.

4.3.2 Hierarchical policy and Q-functions modeling

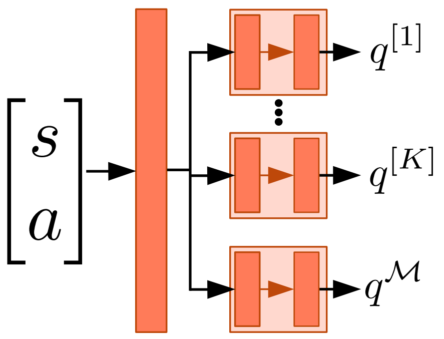
The composable tasks in $\mathcal{T}$ are formulated as independent RL problems, and thus the corresponding policies are also independent. In this line, the mean $\mathbf{m}^{[k]}$ and covariance matrix $\mathbf{C}^{[k]}$ [1] that describe a composable Gaussian policy can be obtained from an independent NN. Nevertheless, we can exploit the assumption that all the tasks in $\mathcal{T}$ share the same state space, and therefore use layers shared across all the policies for obtaining features that can be used in all the policies. The parameters of the resulting NN policy are denoted by $\boldsymbol{\theta}^{\mathcal{T}}$, and include both the parameters of each NN policy and the shared layers.

Moreover, the function $\mathtt{h}$ required for obtaining the state-dependent activation vectors $\{\mathbf{w}^{[k]}\}_1^K = \mathtt{h}(s)$, can also be modeled by an NN with parameters $\boldsymbol{\theta}^w$. These are included in the previously described NN and thus also exploit the features obtained in the shared layers. Therefore, the whole hierarchical policy, depicted in Figure 4.1a, is parameterized by an NN with parameters $\boldsymbol{\theta} = [\boldsymbol{\theta}^{\mathcal{T}} \quad \boldsymbol{\theta}^w]$.

In the same way, an NN with an architecture similar to the hierarchical policy can be used to model the Q-functions of both the composable policies and compound policy. Therefore, the resulting NN, denoted by $Q_{\boldsymbol{\phi}}$ and depicted in Figure 4.1b, has parameters $\boldsymbol{\phi} = [\boldsymbol{\phi}^{\mathcal{T}} \quad \boldsymbol{\phi}^{\mathcal{M}}]$.



(a) Hierarchical policy



(b) Q-value function

Figure 4.1 *Policy and Q-value function neural networks. (a)* the first layer is shared among all the modules, the modules for the composable policies are shown in blue and the module that outputs the activation weights is depicted in purple. The outputs of all these modules are then composed in $\mathtt{f}$ by using one of the alternatives described in section 4.3.1. *(b)* the first layer is also shared among all the modules, and then each module outputs the corresponding approximated Q-value.

[1] More specifically a vector of log standard deviations.

4.4 Simultaneous learning and composition of modular maximum entropy policies

Most methods learn the composable tasks one at a time, and later, the compound task. This procedure is not scalable as all the experience collected during each learning process is only used for that specific process. Also, it is not possible to start learning more complex tasks unless all the composable policies have been successfully learned. This problem is the result of single-task RL formulations where each policy is directly reinforced with its corresponding reward signal. The method proposed in this section is based on the idea that a single stream of experience can be used to improve not only the policy that is generating the behavior but also, indirectly, many other policies. Similar to [88] and [92], the proposed method assumes that the robot receives, at each time step, the rewards for different tasks, and each reward has an assigned policy that tries to maximize its corresponding return G by using the same collection of state-action pairs. Therefore, the motor skills defined by the different composable policies are obtained indirectly from a multi-task formulation and off-policy interaction data collected with the compound policy.

4.4.1 Off-Policy multi-task policy search

The sets of composable policies Π and activation vectors $\mathcal{W}$ required for a compound policy $\pi^{\mathcal{M}}$ to solve task $\mathcal{M}$, can be learned simultaneously. To do so, let us assume that, at each time step, the robot receives a *stream of rewards* $\mathbf{r}_t = [r_t^{[1]} \dots r_t^{[K]} \; r_t^{\mathcal{M}}]^\intercal$, that is, a vector whose components are the reward $r_t^{\mathcal{M}}$ of the compound task $\mathcal{M}$ and the reward $r_t^{[k]}$ of each composable task in $\mathcal{T}$. Moreover, the method considers that the *behavior* or *intentional* policy, i.e. the policy followed by the robot to interact with the environment, is always the compound policy $\pi^{\mathcal{M}}$. The experience at each time step $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$ is collected in the dataset $\mathcal{D}$, and subsequently used for improving compound and composable policies. As a result, the data in $\mathcal{D}$ is off-policy experience for the composable policies because it is generated from a different policy [26]. Thereby, the composable policies (from now on *unintentional* policies [88]) are target policies in the off-policy setting.

Thus, by considering the policy $\pi_{\boldsymbol{\theta}}$ (see section 4.3.2) parameterized by $\boldsymbol{\theta} = [\boldsymbol{\theta}^{\mathcal{T}} \quad \boldsymbol{\theta}^{w}]$, the optimal parameters $\boldsymbol{\theta}^*$ are those that optimize the objective

$$J_\pi(\boldsymbol{\theta}) = J_\pi(\boldsymbol{\theta}^w; \mathcal{M}) + \sum_{k=1}^{K} J_\pi(\boldsymbol{\theta}^{\mathcal{T}}; \mathcal{T}^{[k]}), \tag{4.3}$$

where $J_{\pi}(\boldsymbol{\theta}^{\mathcal{T}};\mathcal{T}^{[k]})$ denotes the performance criterion of the composable policy $\pi_{\boldsymbol{\theta}}^{[k]}$ in task $\mathcal{T}^{[k]}$, and $J_{\pi}(\boldsymbol{\theta}^{w};\mathcal{M})$ the performance criterion of the compound policy $\pi_{\boldsymbol{\theta}}^{\mathcal{M}}$ in task $\mathcal{M}$.

4.4.2 Multi-task Soft Actor-Critic

As mentioned in section 4.2.1, the maximum entropy objective incentives exploration, which is critical for the introduced method as the composable policies are learned unintentionally and their influence in the sampling process is indirect. Thus, each policy seeks to optimize the maximum entropy objective

$$J(\pi^{[j]}) = \sum_{t=0}^{\infty} \mathbb{E}_{\pi^{[j]}} \left[\gamma^{t} \left(r^{[j]}(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}(\pi^{[j]}(\cdot|\mathbf{s}_t)) \right) \right] \tag{4.4}$$

where $r^{[j]}$ is the reward function of the corresponding task $j \in (\mathcal{T} \cup \{\mathcal{M}\})$.

Considering the SAC algorithm described in section 4.2.2, the learning process for all the aforementioned policies is an alternating procedure of policy evaluation, where the value function is computed for all the policies, and policy update, where the policies are improved with their corresponding value functions. Therefore, at each time step, the parameterized Q-function $Q_{\boldsymbol{\phi}}$ optimizes the soft mean-squared bellman error of all the policies

$$J_Q(\boldsymbol{\phi}) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{D}} \left[\frac{1}{2} \sum_{j} \left(Q_{\boldsymbol{\phi}}^{[j]}(\mathbf{s}_t, \mathbf{a}_t) - \left(r^{[j]}(\mathbf{s}_t, \mathbf{a}_t) + \gamma \mathbb{E}_{\mathbf{s}_{t+1}\sim p_s}[V_{\bar{\boldsymbol{\phi}}}^{[j]}(\mathbf{s}_{t+1})] \right) \right)^2 \right] \tag{4.5}$$

for all the tasks $j \in (\mathcal{T} \cup \{\mathcal{M}\})$, where the value function $V_{\bar{\boldsymbol{\phi}}}^{[j]}$ is implicitly parameterized through a target soft Q-function with parameters $\bar{\boldsymbol{\phi}}$ via

$$V_{\bar{\boldsymbol{\phi}}}^{[j]}(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}^{[j]}}[Q_{\bar{\boldsymbol{\phi}}}^{[j]}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi_{\boldsymbol{\theta}}^{[j]}(\mathbf{a}_t|\mathbf{s}_t))].$$

On the other hand, the components of (4.3) for the policy improvement step of the parameterized policy $\pi_{\boldsymbol{\theta}}$ are defined as

$$J_{\pi}(\boldsymbol{\theta}^{\mathcal{T}};\mathcal{T}^{[k]}) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}^{[k]}} \left[Q_{\boldsymbol{\phi}}^{[k]}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi_{\boldsymbol{\theta}}^{[k]}(\mathbf{a}_t|\mathbf{s}_t)) \right] \right] \tag{4.6}$$

for each composable task $\mathcal{T}^{[k]}$ in $\mathcal{T}$, and

$$J_{\pi}(\boldsymbol{\theta}^{w};\mathcal{M}) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}} \left[\mathbb{E}_{\mathbf{a}_t\sim\pi_{\boldsymbol{\theta}}^{\mathcal{M}}} \left[Q_{\boldsymbol{\phi}}^{\mathcal{M}}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log(\pi_{\boldsymbol{\theta}}^{\mathcal{M}}(\mathbf{a}_t|\mathbf{s}_t)) \right] \right] \tag{4.7}$$

Algorithm 1 HIU-SAC

```
1: Initialize target network weights: $\bar{\boldsymbol{\phi}}_i \leftarrow \boldsymbol{\phi}_i$ for $i \in \{1,2\}$
2: Initialize an empty replay memory $\mathcal{D} \leftarrow \emptyset$
3: for each iteration do
4:     for each interaction step do
5:         Sample compound action $\mathbf{a}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$
6:         Sample transition from the environment: $\mathbf{s}_{t+1} \sim p_s(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
7:         Store the interaction data in the replay memory: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})\}$
8:     end for
9:     for each gradient step do
10:        Update $Q$-networks parameters: $\boldsymbol{\phi}_i \leftarrow \lambda_Q \hat{\nabla} J_Q(\boldsymbol{\phi}_i)$ for $i \in \{1,2\}$
11:        Update composable policies parameters: $\boldsymbol{\theta}^{\mathcal{T}} \leftarrow \lambda_\pi \hat{\nabla} J_\pi(\boldsymbol{\theta}^{\mathcal{T}}; \mathcal{T})$
12:        Update compound policy parameters: $\boldsymbol{\theta}^{\mathcal{M}} \leftarrow \lambda_\pi \hat{\nabla} J_\pi(\boldsymbol{\theta}^{\mathcal{M}}; \mathcal{M})$
13:        Update temperature parameters: $\alpha^{[j]} \leftarrow \lambda_\alpha \hat{\nabla} J_\alpha(\alpha^{[j]})$ for $j \in (\mathcal{T} \cup \{\mathcal{M}\})$
14:        Update target Q-networks weights: $\bar{\boldsymbol{\phi}}_i \leftarrow \rho \boldsymbol{\phi}_i + (1-\rho)\bar{\boldsymbol{\phi}}_i$ for $i \in \{1,2\}$
15:    end for
16: end for
```

for the composable task $\mathcal{M}$. Note that the parameters $\boldsymbol{\theta}^w$ required for modeling the activation vectors in $\mathcal{W}$ are the only ones updated in the compound policy because, as discussed in [92], there is no guarantee to preserve the unintentional policies. As a consequence, the proposed algorithm improves the parameterized hierarchical policy $\pi_{\boldsymbol{\theta}}$ in a two-step process with random minibatches from $\mathcal{D}$. First, by optimizing $\boldsymbol{\theta}^{\mathcal{T}}$ with (4.6) for all the composable tasks. And second, by fixing $\boldsymbol{\theta}^{\mathcal{T}}$ and optimizing $\boldsymbol{\theta}^{\mathcal{M}}$ through (4.7).

As suggested in [97], the practical algorithm includes two soft Q-function NNs with parameters $\boldsymbol{\phi}_i$ trained independently to optimize (4.5). Furthermore, the algorithm includes a step to calculate α automatically by optimizing

$$J(\alpha^{[j]}) = \mathbb{E}_{\mathbf{a}_t \sim \pi^{[j]}} \left[-\alpha \log(\pi_{\boldsymbol{\theta}}^{[j]}(\mathbf{a}_t|\mathbf{s}_t)) + \alpha \bar{\mathcal{H}}^{[j]} \right]$$

for $j \in (\mathcal{T} \cup \{\mathcal{M}\})$. Therefore, the whole training process for both composable and compound policies with HIU is summarized in Algorithm 1.

4.5 Experiments

In order to analyze the proposed approach, several experiments were conducted in four robotic tasks that can be intuitively decomposed into simpler tasks. The goal of these experiments is to evaluate if the proposed approach *1)* solves an RL problem with a policy that reuses a set of composable policies, and at the same time, *2)* obtains composable policies with performance similar to dedicated single-task policies.

4.5.1 Tasks description

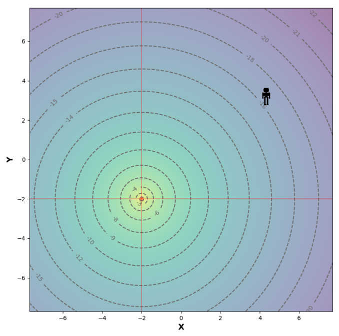
In the first environment, shown in Figure 4.2a, the agent is a 2D point particle that has to reach the position $(-2,-2)$. The state of this environment is continuous and defined by the position (x,y) of the particle, and the control actions are its velocities $(\dot{x},\dot{y})$, then $\mathrm{D}_{\mathcal{S}} = \mathrm{D}_{\mathcal{A}} = 2$. The initial position of the particle is sampled from a spherical Gaussian distribution centered in the position $(4,4)$. This task can be naturally decomposed into two composable tasks, namely, reaching the position -2 in the x coordinate, and reaching the position -2 in the y coordinate. Therefore, the compound policy to reach $(-2,-2)$ has to combine the corresponding composable policies.

The second and third environments correspond to a 3-DoF planar manipulator simulated in Pybullet [98], and whose control actions are joint torques, then $\mathrm{D}_{\mathcal{A}} = 3$. The second environment, shown in Figure 4.2b, requires the robot to reach a random goal pose and is described by a state composed of the joint positions and velocities of the robot, and the relative position of the robot end-effector w.r.t the goal, then, $\mathrm{D}_{\mathcal{S}} = 8$. The third environment, displayed in Figure 4.2c, requires the manipulator to reach a cylinder and push it to a target location. In addition to the joint positions and velocities, the state also includes the positions of the end-effector, the cylinder and the goal, then $\mathrm{D}_{\mathcal{S}} = 12$.

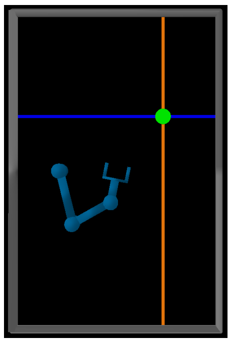
Finally, the proposed approach is also tested in a more complex task, where a simulated CENTAURO robot [72] has to reach a target 3D pose while balancing an object with a tray, as depicted in Figure 4.2d. The tray is firmly attached to the robot hand but not the cylinder. The control actions are the task joint torques of the right arm, then $\mathrm{D}_{\mathcal{A}} = 7$. Note that if a zero-torque control action was applied to the joints, the object would fall by its weight. The state in this scenario is composed of the arm joint positions and velocities, pose errors and rate of change of the errors between the target pose and the center of the tray, and between the desired pose of the cylinder in the tray and its current pose, then $\mathrm{D}_{\mathcal{S}} = 38$.

4.5.2 Robot learning details

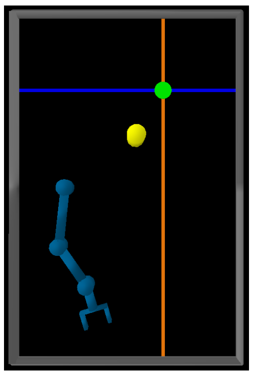
The NN models proposed in section 4.3.2 are used for learning the four tasks above described. The same architecture is used in all the experiments, namely, NNs with ReLU nonlinearities in the hidden nodes and none in the outputs. However, the number of nodes depends on the task, as summarized in Table 4.1.



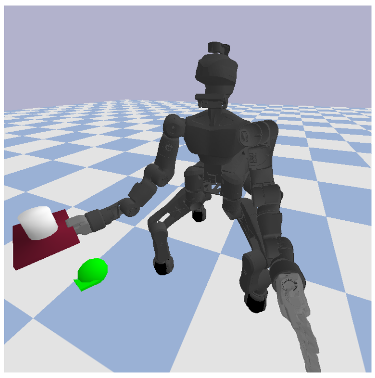
(a)



(b)



(c)



(d)

Figure 4.2 *Experimental scenarios:* (a) 2D particle that reaches a fixed goal position. (b) 3-DoF planar manipulator reaching a random 2D goal position (green circle). (c) 3-DoF planar manipulator that should push the randomly-placed yellow cylinder to a varying 2D goal position (green circle). (d) CENTAURO robot that simultaneously balances a tray with a cylinder and moves to a random 3D goal pose.

The tasks are learned with the algorithm proposed in section 4.4 and the following hyperparameters: Adam optimizer with learning rates $\lambda_Q = \lambda_\pi = \lambda_\alpha = 3 \cdot 10^{-4}$, target smoothing coefficient $\rho = 5 \cdot 10^{-3}$, and discount factor $\gamma = 0.99$. The NNs are trained using stochastic gradient descent with batches sampled from $\mathcal{D}$ after an interaction with the environment. The entropy target $\bar{\mathcal{H}}^{[j]}$ is the same for both composable and compound policies, however its value varies for each task. These values and the replay buffer size $\mathcal{D}$ for each environment are also shown in Table 4.1.

The two composition strategies described in section 4.3 are denoted with HIUSAC. The alternative that considers (4.1) is denoted by HIUSAC-1, while the solution using (4.2) is denoted by HIUSAC-2. For comparison purposes, the SAC algorithm was used to learn both compound and composable policies in a single-task RL formulation.

	2D particle	*3DoF-reacher*	*3DoF-pusher*	*Centauro-tray*
Units per layer	64	128	128	256
Training steps	$1.5\cdot10^4$	$1.5\cdot10^5$	$1.5\cdot10^5$	$1.5\cdot10^6$
Size $\mathcal{D}$	$5\cdot10^6$	$5\cdot10^6$	$5\cdot10^6$	$1\cdot10^7$
Size Minibatch	64	256	256	256
$\bar{\mathcal{H}}$	0	1	1	1

Table 4.1 Environment-specific hyperparameters

4.5.3 Results

Figure 4.3 shows the learning curves of the composable policies obtained with both HIUSAC-1 and HIUSAC-2 for the 2D particle environment. The achieved performance is similar to that obtained directly in the compound task with the SAC algorithm. The approximated soft Q-values for the velocities (actions) given some specific positions of the particle (state) are depicted in Figure 4.4. Notice how the actions and soft Q-values vary as a function of the position, capturing the specifications of their respective tasks. This is a remarkable result as the composable policies were learned unintentionally with off-policy experience collected only with the compound policy.

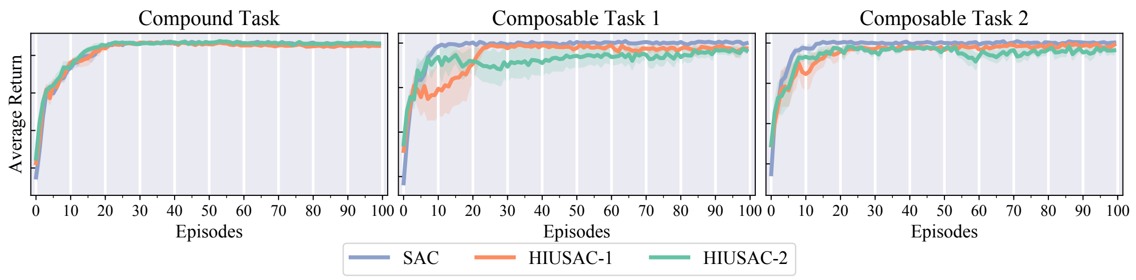


Figure 4.3 *Learning process for the navigation task of a 2D point particle:* SAC denotes the compound and composable policies obtained with the SAC algorithm in single-task formulations. The policies obtained with the algorithm proposed in section 4.3 are denoted by HIUSAC-1 and HIUSAC-2, with the former considering (4.1) and the latter using (4.2). The learning curves show that both the compound and composable policies can successfully perform their respective tasks using the proposed hierarchical model, while being competitive with the single-task formulations.

The learning curves of the other three environments are displayed in Figure 4.5. As noted previously, the tasks with the planar manipulator are more complex than the navigation of the 2D particle because the action space, i.e. joint torques, influence directly in the task of reaching the x position and the task of reaching the y position. Therefore, it is more

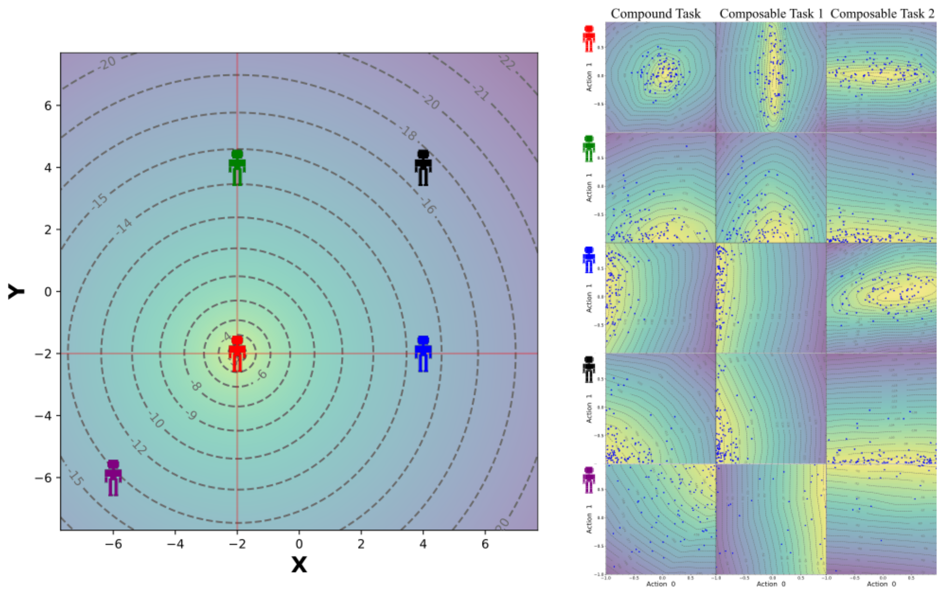
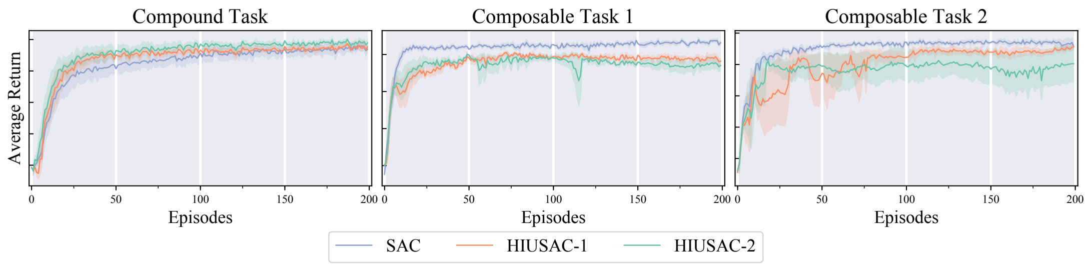


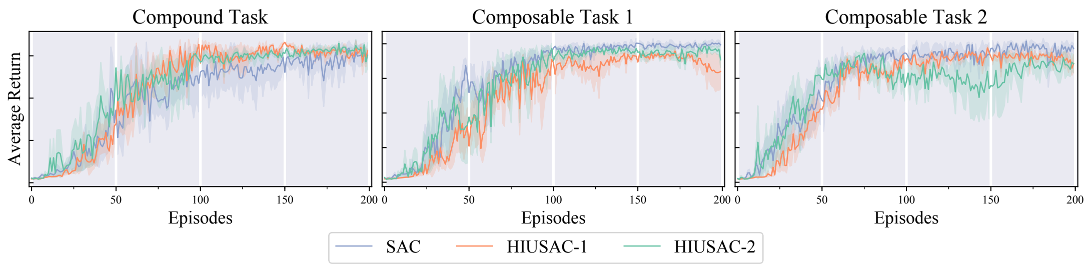
Figure 4.4 *Soft Q-values obtained in the navigation task of a 2D point particle:* The soft Q-values of the compound and composable policies are depicted as contour plots and show the values for the actions in five different positions. Blue points denote some actions sampled from these policies, and they seek to move the point particle to their respective target.

difficult to assign proper activation vectors for the respective composable policies. However, both HIUSAC-1 and HIUSAC-2 obtained successful composable and compound policies, all of them with a performance similar to the policies obtained with the SAC algorithm in single-task RL formulations. However, as we can notice in the composable task 2 for the reaching environment (Figure 4.5a), both alternatives require more iterations to converge.

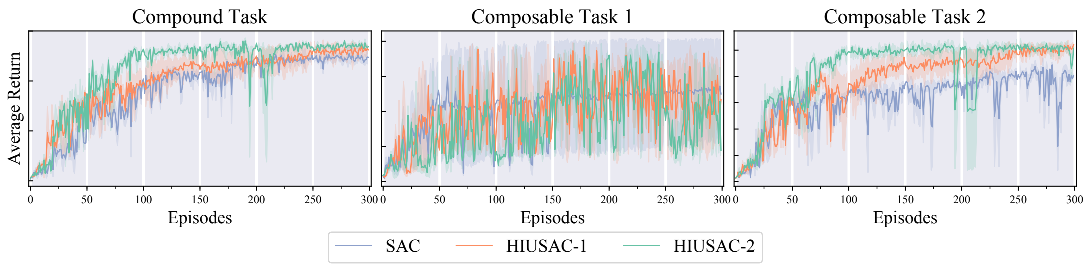
Finally, the results obtained for the task carried out by the CENTAURO robot are reported in Figure 4.5c. In this case, the composable policy converges faster and results in higher average returns when compared to the policy obtained in the single-task formulation. This results demonstrate how the complexity of one task can be solved with a collection of simpler subtasks by exploiting a hierarchical off-policy formulation. An interesting result is that the performance of the composable policies for task 2 exceeds that of their single-task counterparts. A possible explanation for this is that the compound policy explores better the environment and therefore the collected experience contains more meaningful information than the one obtained in the single-task RL formulations. Between HIUSAC-1 and HIUSAC-2, the latter converges faster and results in higher average returns in the compound task and also the composable ones.



(a) Reaching Environment



(b) Pushing Environment



(c) CENTAURO Environment

Figure 4.5 *Learning curves for the compound task and the composable tasks for the reaching, pushing and CENTAURO tasks.* The figures show the average return for the proposed approach with the two composition strategies described in section 4.3. The method that considers (4.1) is denoted by HIUSAC-1. On the other hand, the method that considers (4.2) is denoted by HIUSAC-2. The resulting policies obtained with both alternatives are compared with the ones obtained from single-task RL formulations with the SAC algorithm.

4.6 Challenges

In order to obtain useful data for all the composable policies, the algorithm proposed in this chapter was built on a maximum entropy RL framework to favor exploration during the learning process. Choosing the temperature parameters for the maximum entropy RL objective is challenging for the compound policy because its stochasticity is determined by the activation vectors and the stochasticity of all the composable policies. As a result, high temperature values favors higher entropy policies and the compound policy will show preference for composable policies with high stochasticity. However, this preference is made to the detriment of preferring policies with good performance but low entropy. The automatic entropy adjustment strategy proposed in [97] addresses this problem, however, in this case, the problem is to choose the minimum expected entropy for both composable and compound policies. This value was easier to set for the simple environments of the experiments, but more challenging for the complex ones. Therefore, future work could be focused in developing a mechanism that obtains this value automatically based on the performance of both compound and composable policies.

On the other hand, in this chapter, the action value functions of the composable policies are only used in the policy evaluation step of these policies. However, these models capture the performance of the policies in their respective tasks, and therefore they are important sources of information that could be considered by the compound policy.

Finally, an important motivation for task decomposition is reusing the composable policies in new tasks. Future work will analyze the behavior of the composable policies when they are reused in different compound tasks. It is important to know how the performance of new compound policies is affected by composable policies obtained in different contexts. Moreover, the experiments conducted in this chapter were focused in tasks that are decomposed in two subtasks. Future work will consider tasks that could be decomposed in more than two subtasks and tasks where the components of the action vector can be assigned completely to specific tasks, e.g. bimanual tasks.

4.7 Summary

This chapter detailed a hierarchical RL approach for tasks that can be decomposed into a collection of subtasks that require to be performed concurrently. The Gaussian policies corresponding to these subtasks are combined using a set of activation vectors. These activation vectors permit to consider concurrently actions sampled from all the low-level policies and preferences among specific components. Furthermore, two methods were proposed to obtain a compound policy that is also Gaussian and a function of the means and covariances matrices of the composable policies.

Moreover, the chapter described an algorithm for learning both compound and composable policies within the same learning process by exploiting the off-policy data generated from the compound policy. Note that populating the replay memory buffer with rich experiences is essential for acquiring multiple skills in an off-policy manner. The composable policies learned unintentionally had similar performance than the policies obtained in single-task formulations only when the compound policy was able to efficiently explore the environment. For this reason, the algorithm was built on a maximum entropy RL framework to favor exploration during the learning process.

Chapter 5

EXPLOITING GOOD AND BAD EXPERIENCES IN POLICY LEARNING

As with any other RL agent, the experiences that a robot obtains while learning a task are not always good or successful. Thus, the resulting undesired behavior is irrational with respect to the performance measure that defines the objective of the task. As a consequence, most of the robot learning formulations will increase the probability of occurrence of high return trajectories and give only low probability to the failures by updating the robot's policy [10, 42]. However, reducing the likelihood of actions that induce the undesirable rollouts does not necessarily prevent their repetition. For example, an aggressive exploration, a flawed definition of the robot/environment state, the existence of limited useful data or the stochasticity of the environment may lead the robot to fail again and therefore generate bad or undesirable rollouts.

Furthermore, complex nonlinear models, such as deep neural networks (NN), allow the representation of policies that induce complex behaviors and permit to work directly with sensory input data [74, 75, 99]. Nonetheless, the prohibitively amount of data required when learning these models has been significantly reduced with guided policy search (GPS) algorithms. GPS methods are data-efficient because they transform the policy search problem into supervised learning, where the training data is generated by a computational teacher that produces data that is best suited for training the final policy.

Failures can be even more critical in GPS, because the guiding policies seek to optimize an expected surrogate return which includes not only the task-related reward, but also a term that encourages the guiding policies to resemble the complex global policy. The latter term may lead the guiding policies to produce undesired events that generate a sudden decrement in the reward values, which we necessarily want to avoid in the following iterations. Our desire to avoid these failures can, to a certain extent, be captured by a well-designed reward function. However, as in the case of mirror descent guided policy search (MDGPS) [100], the change of the guiding policies is limited by a Kullback-Leibler (KL) divergence constraint, and therefore samples generated by these updated policies may produce the same undesired failures of previous iterations. Additionally, the supervision step in GPS uses all the trajectory samples that were generated by the guiding policies, therefore fitting the global policy to trajectories that, in fact, we are trying to avoid. Consequently, next guiding policies will be constrained to be similar to a global policy that was trained with flawed samples.

This chapter proposes an alternative approach for reducing the occurrence of failures during the learning process by modeling the undesirable behavior and making it explicitly influence the robot's policy. More specifically, the interactions that result in failures, and that are usually overlooked for the robot's policy, are explicitly considered by modeling policies, so-called bad policies, that can reproduce them. Similarly, good policies are constructed from only successful executions, and along the bad policies, influence the skill acquisition process as *dualist constraints*. Thus, the robot's policy is not directly updated from the performance of the resulted behavior. Instead, it is additionally updated according to its similarity with the good policies, that induce successful executions, and dissimilarity with the bad policies, that induce failures. In addition, neither performance measure and good-bad policies are considered directly by the policy, which is modeled with a NN. Instead, it is proposed an extension of GPS that considers trajectories optimized with the dualist constraints. The conducted experiments suggest that NN policies guided by trajectory distributions optimized with this method reduce the failures during the policy exploration phase, and therefore encourage safer interactions.

5.1 Related Work

The possibility to use robust algorithms and simple local policies with few parameters has made GPS a popular framework to learn complex policies. Simple local policies are employed as computational teachers that generate guiding distributions for a nonlinear global policy. Such simple policies are usually trajectory-centric representations such as splines, dynamic movement primitives [101] and time-varying linear-Gaussian (TVLG) controllers. The latter are popular in stochastic optimal control and various trajectory optimization methods, such as the iterative linear quadratic Gaussian (iLQG) algorithm [102, 103]. Typically the algorithms used to optimize TVLG controllers assume a known (or iteratively learned) dynamics model [23, 104]. In this chapter, TVLG controllers are employed to represent the simple policies but also the trajectory distributions constructed from both successful and failed executions.

No GPS algorithm exploits the existence of poorly performing samples. For example, in PI-GPS [105] the samples with high cost-to-go (low return) values are practically ignored if better samples are obtained in the same iteration because of the assigned low-probability scores. As a consequence, the high-cost (low-reward) samples are barely considered in subsequent policy updates. Note that giving this treatment to failed executions may generate policy updates that still lead to failures in the next few iterations. Nevertheless, learning from failures is a promising approach to discover successful and safer ways to accomplish tasks. For example, if a dataset of negative/undesired samples is explicitly considered, it is possible to exclude regions in the parameter space that lead to failures by favoring regions that conduce to successful executions [106, 107].

Notice that in inverse reinforcement learning (IRL), considering negative state-action trajectories generates more accurate reward functions than methods relying on positive trajectories exclusively. Shiarlis et al. [108], for example, included negative demonstrations into the optimization of a maximum-causal-entropy IRL method. In such a way, the method obtained, in less iterations, linear rewards functions that generalized better even when the successful and failed demonstrations were contrasting, overlapping, or complementary. Similarly, Choi et al. [109] proposed to use a Gaussian process to represent a non-linear reward function whose kernel moved the prediction close to positive samples and away from negative ones. Beyond the fact that the approach detailed in this chapter addresses a policy search problem, the main difference of this approach with respect to the aforementioned techniques is that the agent does not have access to previous datasets of positive and negative trajectories. Instead, both types of trajectories are obtained during its interaction with the environment and latter classified as good or bad samples based on their return.

The method proposed in this chapter is inspired by the dual relative entropy policy search (DREPS) [24] algorithm, a generalization of REPS [110] that takes into account both good and bad samples when computing a policy. In DREPS, a closed form update of the parameters is obtained from the Lagrangian of an optimization problem that bounds the KL divergence between the new policy and precomputed low- and high-performance clusters of parameters. In this chapter the robot policy is represented by neural networks, models with high dimensional parameter spaces where constructing the clusters is difficult and applying REPS unfeasible. Instead, the good and bad samples are used to construct trajectory distributions which are latter considered as constraints for simple low-dimensional policies. In consequence, there are several differences with DREPS, the most important one is that the good and bad interactions are considered indirectly by the robot policy, because the guiding policies are the only ones that take them into account. Second, the algorithm in this chapter is formulated in a step-based policy search setting, which means that the method has notion of state-space and sequential decisions. Finally, the proposed algorithm is formulated on a model-based RL setting, where a (local) dynamics model has to be learned before carrying out the policy optimization step.

5.2 Preliminaries

A robot behavior is generated by a parametrized stochastic policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}|\mathbf{s})$, a conditional distribution over action $\mathbf{a}$ given the state $\mathbf{s}$. In this chapter, episodic tasks are considered, as a consequence, the policy and the transition dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$ induce a trajectory $\tau = \{\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T\}$ with probability

$$\pi_{\boldsymbol{\theta}}(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t),$$

where $p(\mathbf{s}_1)$ is the initial state distribution. Thus, the goal of policy search (PS) is to optimize the parameters $\boldsymbol{\theta}$ with respect to the expected accumulated reward or return

$$\mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}(\cdot)}\left[G(\tau)\right] = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)\right].$$

5.2.1 Model-based trajectory-centric reinforcement learning

The iterative linear quadratic Gaussian algorithm (iLQG) [102, 103] is an efficient shooting method [1] in trajectory optimization. Unlike differential dynamic programming (DDP) [111] which considers higher-order terms, iLQG only uses the first derivative of the discrete-time dynamics, thus allowing a faster dynamics evaluation that outweighs the decrease in performance. iLQG considers a quadratic expansion of the reward $r(\mathbf{s}_t,\mathbf{a}_t)$ around a nominal trajectory $\bar{\tau} = \{\bar{\mathbf{s}}_1,\bar{\mathbf{a}}_1,\ldots,\bar{\mathbf{s}}_T,\bar{\mathbf{a}}_T\}$, and a local linear-Gaussian approximation of the dynamics $\mathcal{N}(f_{\mathbf{s}t}\mathbf{s}_t + f_{\mathbf{a}t}\mathbf{a}_t + f_{ct},\mathbf{F}_t)$ with a covariance $\mathbf{F}_t$, and a mean given by the gradients $f_{\mathbf{s}t}$ and $f_{\mathbf{a}t}$, and a constant term f_{ct}. Under these assumptions, the first and second derivatives of the

[1] also called indirect trajectory optimization.

action-value function $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ and the state-value function $V^{\pi}(\mathbf{s}_t)$ are:

$$
\begin{aligned}
Q_{\mathbf{sa}t} &= \mathbf{r}_{\mathbf{sa}t} + f_{\mathbf{sa}t}^{\top} V_{\mathbf{s}t+1} \\
Q_{\mathbf{sa},\mathbf{sa}t} &= \mathbf{r}_{\mathbf{sa},\mathbf{sa}t} + f_{\mathbf{sa}t}^{\top} V_{\mathbf{s},\mathbf{s}t+1} f_{\mathbf{sa}t} \\
V_{\mathbf{s}t} &= Q_{\mathbf{s}t} - Q_{\mathbf{a},\mathbf{s}t}^{\top} Q_{\mathbf{a},\mathbf{a}t}^{-1} Q_{\mathbf{a}t} \\
V_{\mathbf{s},\mathbf{s}t} &= Q_{\mathbf{s},\mathbf{s}t} - Q_{\mathbf{a},\mathbf{s}t}^{\top} Q_{\mathbf{a},\mathbf{a}t}^{-1} Q_{\mathbf{a},\mathbf{s}t}
\end{aligned}
$$

where the derivatives are denoted by subscripts, so for example, $r_{\mathbf{sa}t}$ is the gradient of the reward at time step t with respect to $[\mathbf{s}, \mathbf{a}]^{\top}$ and $r_{\mathbf{sa},\mathbf{sa}t}$ the Hessian. The optimal linear control law that maximizes $Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)$ is $g(\mathbf{s}_t) = \mathbf{K}_t(\mathbf{s}_t - \bar{\mathbf{s}}_t) + \mathbf{k}_t + \bar{\mathbf{a}}_t$, where $\mathbf{K}_t = -Q_{\mathbf{a},\mathbf{a}t}^{-1} Q_{\mathbf{a},\mathbf{s}t}$ and $\mathbf{k}_t = -Q_{\mathbf{a},\mathbf{a}t}^{-1} Q_{\mathbf{a}t}$.

A Gaussian trajectory distribution $p(\tau)$ can be obtained by considering a linear-Gaussian controller $p(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t, \mathbf{C}_t)$. The mean of this distribution is given by the above deterministic solution assuming, for notational convenience, that the nominal states and actions are zero. The covariance $\mathbf{C}_t$ is proportional to the curvature of the Q-function, $\mathbf{C}_t = Q_{\mathbf{a},\mathbf{a}t}^{-1}$. Note that this linear-Gaussian controller also optimizes the maximum entropy objective as shown in [23], which is formulated as:

$$
\begin{aligned}
p(\tau) \leftarrow \underset{p(\tau)\in\mathcal{N}(\tau)}{\operatorname{argmax}}\ & \mathbb{E}_{p(\tau)}[G(\tau)] + \mathcal{H}(p(\tau)) \\
s.t.\ & p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) = \mathcal{N}(f_{\mathbf{s}t}\mathbf{s}_t + f_{\mathbf{a}t}\mathbf{a}_t + f_{ct}, \mathbf{F}_t)
\end{aligned}
$$

When the dynamics is unknown, a distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ can be estimated around the trajectories sampled from the real system under the previous linear-Gaussian controller, denoted by $\hat{p}(\mathbf{a}_t|\mathbf{s}_t)$. To prevent the dynamic programming pass in the iLQR from drastically modifying the new controller, which makes the local dynamics invalid around the new trajectory distribution, the policy update should be constrained. Levine and Abbeel [23] proposed to include a KL divergence constraint on the previous trajectory distribution $\hat{p}(\tau)$, as follows:

$$
\max_{p(\tau)} \mathbb{E}_{p(\tau)}[G(\tau)] \quad \text{s.t.} \quad D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau)) \leq \varepsilon, \tag{5.1}
$$

where ε denotes the maximal information loss, and the dynamics constraint has been omitted for clarity. The Lagrangian of (5.1) is

$$
\mathcal{L}(p(\tau), \eta) = \mathbb{E}_{p(\tau)}[G(\tau)] + \eta[\varepsilon - D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau))],
$$

and since $p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t) = \hat{p}(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$, the resulting Lagrangian becomes

$$\begin{aligned}\mathcal{L}(p(\tau),\eta) = &\left[\sum_{t=1}^{T} \mathbb{E}_{p(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t) + \eta \log \hat{p}(\mathbf{s}_t,\mathbf{a}_t)]\right] \\ &+ \eta \mathcal{H}(p(\tau)) + \eta\varepsilon \end{aligned} \tag{5.2}$$

The above problem is solved in [23] by dual gradient descent, alternating between a dynamic programming pass to optimize the Lagrangian with respect to $p(\tau)$, and adjusting η according to the amount of constraint violation [2].

5.2.2 Guided policy search algorithms

Despite the important advances in model-based and model-free PS methods [42], their application is generally limited to specific policy representations with less than a hundred parameters, or require a prohibitively amount of interactions with the environment [99]. Instead of optimizing the parameters directly from the expected return, GPS methods transform the policy search problem into supervised learning, where the training set is generated by a computational teacher, optimized by either simple trajectory-centric RL algorithms [75] or complex trajectory optimization methods [112]. In such a way, the convergence of a global policy that maximizes the expected accumulated reward is obtained by solving

$$\max_{\boldsymbol{\theta},p(\tau)} \mathbb{E}_{p(\tau)}[G(\tau)] \quad \text{s.t.} \quad p_i(\mathbf{a}_t|\mathbf{s}_t) = \pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t) \quad \forall t, \forall i, \tag{5.3}$$

meaning that the learning process of the global policy is indeed divided into a domain-specific optimization of *local policies* $p_i(\mathbf{a}_t|\mathbf{s}_t)$, and a supervised phase for the *global policy* $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$ so that it matches the simple policies. When $p_i(\mathbf{a}_t|\mathbf{s}_t)$ is represented by a TVLG controller, the method described in section 5.2.1 can be used.

The constrained maximization problem in (5.3) guarantees that the global policy maximizes the expected return at convergence, but barely focus on the robot behavior in intermediate iterations. This limitation is solved by MDGPS [100] which, under linearity and convexity assumptions, approximates a local TVLG controller $\bar{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$ to the global policy

[2]The derivation of [23] is actually for the Lagrangian of a minimization cost problem. However, the Lagrangian of the maximization reward problem proposed here is equivalent when the formulation of iLQG described in this section is considered.

$\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$ by reformulating the local policy constraint in (5.1) as:

$$\max_{p(\tau)} \mathbb{E}_{p(\tau)}[G(\tau)] \quad \text{s.t.} \quad D_{\text{KL}}(p(\tau)||\bar{\pi}_{\boldsymbol{\theta}}(\tau)) \leq \varepsilon, \tag{5.4}$$

where $\bar{\pi}_{\boldsymbol{\theta}}(\tau)$ is the trajectory induced by $\bar{\pi}_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$.

Note that the global policy learned by GPS exclusively focus on behaviors provided by the guiding distributions $p_i(\tau)$, meaning that it ignores completely how these trajectories were obtained. Thus, if we require the complex global policy to consider bad behaviors during the learning process, the local policies updates must explicitly consider the undesired low-return experiences into the optimization problem. In such a way, the policy search is expected to avoid policy parameters that generate unsafe or unsuccessful executions, therefore being less prone to failures.

5.3 Deep reinforcement learning with dualist updates

First of all, let us define $\mathcal{G}$ and $\mathcal{B}$ as the sets representing good and bad experiences, respectively. In a policy search setting, these experiences are encoded by good and bad trajectory probability distributions that are generated from successful (high-return) or failed (low-return) task executions. These trajectory distributions can be explicitly considered in the policy update by including an upper bound on the KL divergence between the new and good trajectory distributions, and a lower bound on the KL divergence between the new and bad trajectory distributions. In this way, similarly to [24], we reformulate the policy update as follows

$$\begin{aligned} \boldsymbol{\theta} = &\operatorname*{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\tau)}[G(\tau)] \\ s.t.\ &D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(\tau)||g_c(\tau)) \leq \chi,\ c \in \mathcal{G} \\ &D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(\tau)||b_d(\tau)) \geq \xi,\ d \in \mathcal{B} \end{aligned} \tag{5.5}$$

where $g_c(\tau)$ are the good trajectory distributions in $\mathcal{G}$, $b_d(\tau)$ are the bad trajectory distributions in $\mathcal{B}$, χ the maximal information loss with respect to the good trajectory distributions and ξ the minimal information loss with respect to the bad trajectory distributions. The interpretation of these dualist constraints is very intuitive: the parameterized policy that considers dualist constraints induces a trajectory distribution that differs from trajectory distributions that generate bad behaviors and whose sampled trajectories resemble previous successful executions.

Note that when the policy is represented by a deep neural network, the optimization process using typical RL methods is infeasible due to the high dimensionality of the policy parameters. Therefore, GPS (described in section 5.2.2) is proposed in order to generate guiding local policies to assist and accelerate the policy search process. In this sense, GPS needs to be reformulated in order to include dualist constraints as explained next.

5.3.1 Model-based (trajectory-centric) RL with dualist updates

Without loss of generality, let us assume that both good and bad behaviors are generated from a single trajectory distribution each. Then, the cardinalities of the good and bad sets are $|\mathcal{G}| = |\mathcal{B}| = 1$. Let us also define good and bad policies as $g(\mathbf{a}_t|\mathbf{s}_t)$ and $b(\mathbf{a}_t|\mathbf{s}_t)$, that induce a good trajectory distribution $g(\tau)$ and a bad trajectory distribution $b(\tau)$, respectively. Therefore, the policy update in PS can be reformulated as

$$\begin{aligned} p(\tau) \leftarrow & \operatorname*{argmax}_{p(\tau)} \mathbb{E}_{p(\tau)}[G(\tau)] \\ s.t.\ & D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau)) \leq \varepsilon \\ & D_{\mathrm{KL}}(p(\tau)||g(\tau)) \leq \chi \\ & D_{\mathrm{KL}}(p(\tau)||b(\tau)) \geq \xi. \end{aligned} \tag{5.6}$$

The Lagrangian of (5.6) is defined as

$$\begin{aligned} \mathcal{L}(p(\tau),\eta,\omega,\nu) =& \mathbb{E}_{p(\tau)}[G(\tau)] + \eta[\varepsilon - D_{\mathrm{KL}}(p(\tau)||\hat{p}(\tau))] \\ & + \omega[\chi - D_{\mathrm{KL}}(p(\tau)||g(\tau))] \\ & + \nu[D_{\mathrm{KL}}(p(\tau)||b(\tau)) - \xi], \end{aligned} \tag{5.7}$$

where η, ω and ν are the Lagrange multipliers controlling the relevance of each inequality constraint in (5.6). Due to the linear-Gaussian dynamics assumption, the maximization of (5.7) with respect to $p(\tau)$ can be written as:

$$\begin{aligned} \max_{p(\tau)} \quad & \mathbb{E}_{p(\tau)}\left[\sum_{t=1}^{T} \mathbb{E}_{p(\mathbf{s}_t,\mathbf{a}_t)}[\tilde{r}(\mathbf{s}_t,\mathbf{a}_t)]\right] + \mathcal{H}(p(\tau)) \\ & + \frac{1}{\eta+\omega-\nu}[\eta\varepsilon + \omega\chi - \nu\xi] \end{aligned} \tag{5.8}$$

Algorithm 2 Dualist GPS

```
1: Initialize local policies $p_i$
2: for iteration $k = 1$ to $K$ do
3:     Run $p_i$ to collect trajectory samples $\mathcal{D}_i = \{\tau_i\}$
4:     Fit linear-Gaussian dynamics $p_i(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ to $\mathcal{D}_i$
5:     Fit linearized global policy $\bar{\pi}_{\boldsymbol{\theta}_i}(\mathbf{a}_t|\mathbf{s}_t)$ to $\mathcal{D}_i$
6:     Update $g(\tau)$ and $b(\tau)$ from $\mathcal{D}_i$
7:     Adjust $\varepsilon$
8:     Optimize $p_i$ from (5.9)
9:     Optimize $\pi_{\boldsymbol{\theta}}$ from (5.13)
10: end for
```

where

$$\tilde{r}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{\eta + \omega - \nu}\left[r(\mathbf{s}_t, \mathbf{a}_t) + \eta \log \hat{p}(\mathbf{a}_t|\mathbf{s}_t) + \omega \log g(\mathbf{a}_t|\mathbf{s}_t) - \nu \log b(\mathbf{a}_t|\mathbf{s}_t)\right].$$

Then, considering this augmented reward $\tilde{r}(\mathbf{s}_t, \mathbf{a}_t)$, the primal problem in (5.6) can also be solved using (5.8) with a process similar to that described in section 5.2.1.

5.3.2 Dualist GPS

Learning a complex nonlinear policy by solving problem (5.5) usually requires a large amount of interactions, where failed (possible dangerous) executions might arise until an acceptable suboptimal policy is obtained. These failures may lead to serious consequences on the robot and the environment it interacts with. Therefore, by reformulating the GPS framework to include dualist constraints, we can reduce not only the interactions required to learn $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$, but also the robot failures during the learning process.

As mentioned in section 5.2.2, a global policy learned by the classical GPS optimizes its parameters based only on the behaviors provided by the guiding distributions $p_i(\tau)$. Thus, using trajectory distributions optimized with the trajectory-centric RL algorithm proposed in the previous section, implies that the global policy $\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)$ considers both good and bad experiences, as outlined in Algorithm 2.

The proposed algorithm employs MDGPS to learn the global policy. Therefore, the optimization problem (5.6) for each local policy i is rewritten as

$$p_i(\tau) \leftarrow \underset{p_i}{\operatorname{argmax}} \mathbb{E}_{p_i(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{5.9}$$

$$s.t.\ D_{\mathrm{KL}}(p_i(\tau) || \bar{\pi}_{\boldsymbol{\theta}_i}(\tau)) \leq \varepsilon \tag{5.10}$$

$$D_{\mathrm{KL}}(p_i(\tau) || g_i(\tau)) \leq \chi \tag{5.11}$$

$$D_{\mathrm{KL}}(p_i(\tau) || b_i(\tau)) \geq \xi \tag{5.12}$$

where, $\bar{\pi}_{\boldsymbol{\theta}_i}(\tau)$ is the trajectory induced by the local TVLG approximation $\bar{\pi}_{\boldsymbol{\theta}_i}(\mathbf{a}_t|\mathbf{s}_t)$. Note that good $g_i(\tau)$ and bad $b_i(\tau)$ distributions are defined for each $p_i(\tau)$, and updated by samples generated by their respective TVLG controller.

The supervision step in MDGPS is the same as the original formulation, meaning that the parameters of the global policy $\pi_{\boldsymbol{\theta}}$ are obtained by:

$$\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{t,i,j} D_{\mathrm{KL}}(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_{t,i,j}) || p_i(\mathbf{a}_t|\mathbf{s}_{t,i,j})) \tag{5.13}$$

where $\mathbf{s}_{t,i,j}$ is the j^{th} sample from $p_i(\mathbf{s}_t)$ acquired by performing $p_i(\mathbf{a}_t|\mathbf{s}_t)$.

5.3.3 Defining good and bad experiences

Classifying a sampled trajectory either as good or bad, clearly involves an assessment based on different, possibly many, criteria. If the learning process is carried out under the supervision of a human, then the human may provide feedback regarding the goodness of every robot execution. But in autonomous settings, our expectations of what we want the robot *to do* and *not to do* are mainly captured by the reward function. The main goal of a robot in an RL problem is to maximize the expected accumulated reward or return $G(\tau)$. For this reason, bad experiences are here defined as undesirable trajectories that have low return. Similarly, good experiences are trajectories with high return. In such a way, at each iteration k, n_g good samples and n_b bad samples are used to update the trajectory distributions in $\mathcal{G}$ and $\mathcal{B}$, respectively. Note that these definitions rely on the fact that the reward function captures not only how the robot behavior should be to perform optimally, but also how it should never be.

As mentioned before, it is assumed that the cardinalities of the good and bad sets are $|\mathcal{G}| = |\mathcal{B}| = 1$. This involves that all the n_g trajectory samples with higher return are used

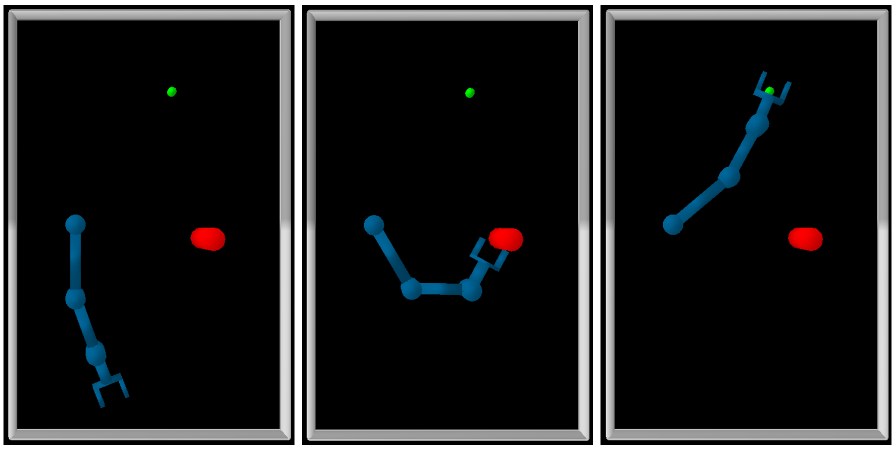


Figure 5.1 *Reaching task of a planar manipulator.* The robot should learn to reach a target Cartesian pose (depicted in green) without touching an obstacle (red cylinder). During the iterative learning process, the robot may collide with the obstacle generating low-return executions that are considered failures. Such failed executions are exploited by dualist GPS to provide safer global policies.

to update a single good trajectory distribution $g(\tau)$ and all the n_b trajectory samples with low return are used to update a single bad trajectory distribution $b(\tau)$. However, there are different ways to construct the distributions $g(\tau)$ and $b(\tau)$, in this chapter they are obtained by following the same method employed in MDGPS to get the local TVLG approximation $\bar{\pi}_{\boldsymbol{\theta}}(\tau)$. Specifically, at each iteration k, a TVLG controller fits the observed good trajectories and another TVLG controller the observed bad trajectories.

5.4 Experiments

The proposed framework was evaluated in two reaching tasks with collision avoidance. Both a simulated 3-DoF planar manipulator and a simulated humanoid robot were required to reach a desired Cartesian pose while avoiding an obstacle that is halfway.

5.4.1 Tasks description

The first reaching task is performed by a 3-DoF planar manipulator as shown in Figure 5.1. The state of the task is defined by $\mathbf{s} \in \mathbb{R}^{12}$, composed of joint positions $\mathbf{q} \in \mathbb{R}^3$, joint velocities $\dot{\mathbf{q}} \in \mathbb{R}^3$, and end-effector pose relative to the target $\mathbf{g} \in \mathbb{R}^3$ and to the obstacle $\mathbf{o} \in \mathbb{R}^3$. The action $\mathbf{a} \in \mathbb{R}^3$ corresponds to the robot torque commands.

The reward function that evaluates the performance of the task execution was defined as:

$$r(\tau) = r(\mathbf{s}_T, \mathbf{a}_T) + \sum_{t=1}^{T-1} r(\mathbf{s}_t, \mathbf{a}_t) \tag{5.14}$$

with final reward

$$r(\mathbf{s}_T, \mathbf{a}_T) = w_1(\gamma_1 + ||\mathbf{g}_T||^2)^{1/2} + w_2||\mathbf{g}_T||^2 + \ w_3 \max(d_{\text{SAFE}} - d(\mathbf{o}_T), 0),$$

and stage reward

$$r(\mathbf{s}_t, \mathbf{a}_t) = w_4(\gamma_2 + ||\mathbf{g}_t||^2)^{1/2} + w_5||\mathbf{g}_t||^2 + w_6 \max(d_{\text{SAFE}} - d(\mathbf{o}_t), 0) + w_7||\mathbf{u}_t||^2$$

where $d(\mathbf{o})$ is the signed distance evaluated on the obstacle pose $\mathbf{o}$. The term $\max(d_{\text{SAFE}} - d(\mathbf{o}), 0)$ is a hinge loss that does not penalize the robot if the distance $\mathbf{o}$ is further than d_{SAFE} and favors collision avoidance behaviors [113]. The values for the different variables in this reward function are depicted in Table 5.1.

The second reaching task was carried out by a simulated CENTAURO robot. In this environment, the humanoid seeks to reach a Cartesian pose while avoiding a cylindric obstacle that is halfway, as shown in Figure 5.2. The state of the task is defined by $\mathbf{s} \in \mathbb{R}^{26}$, composed of joint positions of the right arm $\mathbf{q} \in \mathbb{R}^7$, joint velocities of the right arm $\dot{\mathbf{q}} \in \mathbb{R}^7$, and both position and orientation errors between the right hand of the robot and both the target $\mathbf{g} \in \mathbb{R}^6$ and the obstacle $\mathbf{o} \in \mathbb{R}^6$. The action $\mathbf{a} \in \mathbb{R}^7$ corresponds to the robot right arm task-torque commands. As we can notice, this learning task is more complex than the one in the previous experiment because, first, its state-action space has higher dimensionality, and second, the target is closer to the obstacle which increases the possibility of colliding with the obstacle.

The reward function of the reaching task with CENTAURO is similar than the one performed by the 3DoF planar manipulator, except by the use of a Lorentzian ρ-function [11]. Specifically, the terms $(\gamma_1 + ||\mathbf{g}_T||^2)^{1/2}$ and $(\gamma_2 + ||\mathbf{g}_t||^2)^{1/2}$ are replaced by $\log(\gamma_1 + ||\mathbf{g}_T||^2)$ and $\log(\gamma_2 + ||\mathbf{g}_t||^2)$, respectively. In addition, the new values for the variables in the reward function are depicted in Table 5.1.

The NN global policies for both robots consisted of fully connected feed-forward neural networks with two hidden layers and ReLU nonlinearities. The hidden layers for the 3DoF planar manipulator and CENTAURO had 40 and 64 units respectively. Four Cartesian target and obstacle poses were randomly generated for each robot, therefore four TVLG controllers ($i = 4$) were iteratively optimized as guiding policies.

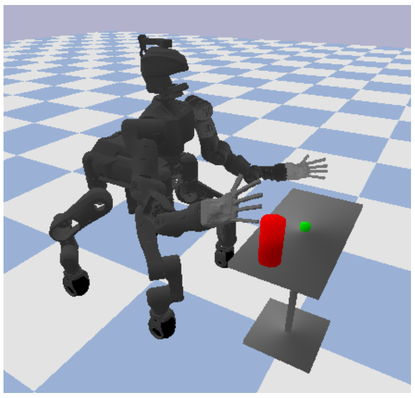


Figure 5.2 *Reaching task of CENTAURO*. The humanoid is requested to learn to reach a desired Cartesian pose (depicted in green) with its right arm without touching the obstacle (red cylinder).

	3DoF planar robot	CENTAURO
$w1$	-1×10^4	-300
$w2$	-50	-300
$w3$	-2×10^6	-500
$w4$	-10	-30
$w5$	-5×10^{-2}	-30
$w6$	-2×10^3	-50
$w7$	-1×10^{-6}	-0.1
γ_1	10^{-10}	10^{-5}
γ_2	10^{-10}	10^{-5}
d_{SAFE}	0.15	0.217

Table 5.1 Values considered in the reward function for the reaching task for the 3DoF planar manipulator and CENTAURO

5.4.2 Results

The first experiments conducted in the reaching task with the 3DoF planar manipulator were focused on analyzing how many times and how fast the robot approached, or even collided, with the obstacle, during the iterative learning process. The experiments aimed at observing the effect of discarding the worst samples in the supervised learning (SL) step of MDGPS, and optimizing the NN parameters to match the remaining trajectory samples. As a result, the NN was trained to match a smaller dataset but composed of non-bad trajectories. This intuitive strategy may be justified by the fact that we require the global policy to not reproduce the trajectories that generated low return. The results of the experiments are depicted in Figure 5.3.

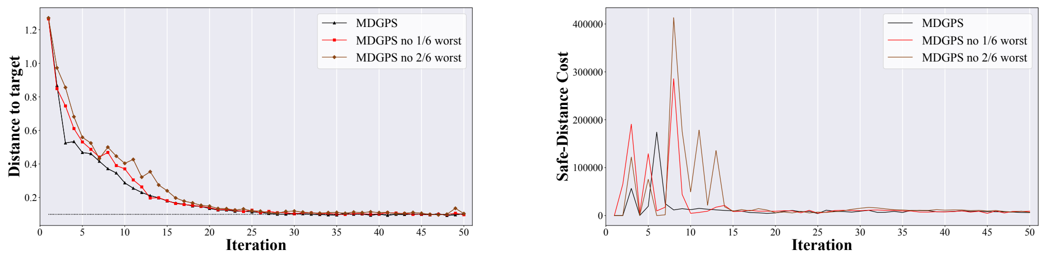


Figure 5.3 *Experiment disregarding worst trajectory samples*. Disregarding the worst samples in the SL step of MDGPS does not avoid the planar robot from failing. *MDGPS* corresponds to the case where all the 6 samples for each local policy are considered. *MDGPS no 1/6 worst*, means that the worst trajectory sample (the one with lowest return) is disregarded. *MDGPS no 2/6 worst* means that the two trajectory samples with lowest return are disregarded.

The right plot in Figure 5.3 displays the accumulated safe-distance cost, $\sum_{t=1}^{T} \max(d_{\text{SAFE}} - d(\mathbf{o}_t), 0)$, for the training conditions, where high values indicate that the resulting trajectory generated a dangerous robot end-effector movement as it was very close to the obstacle. Thus, the higher the value, the worst the trajectory, and therefore the more important it is to avoid trajectory distributions that generate this undesired behavior. Note that despite the negative samples that generated the higher values of this cost were identified and discarded as training data for the SL step, the NN policy (and subsequently the samples from the trajectory distributions that guided it) continued passing near the obstacle. Only after some more iterations the updated trajectory distributions resulted in end-effector movements that avoided the obstacle. On the other hand, the left plot of Figure 5.3 shows that discarding the bad samples makes the robot require more iterations to reach the desired pose, therefore, it has a negative impact on the main objective of the task. This effect may arise as consequence of the reduction of the training dataset used to train the neural network, which deteriorated its performance. Again, only after some more iterations (and no more bad trajectories), the final performance was the same as the standard MDGPS.

Later, three different GPS formulations were compared in the reaching tasks of both robots. The first one did not consider neither (5.11) nor (5.12) constraints, hence corresponding to the standard MDGPS. The second formulation only took into account bad experiences, which means that (5.11) was not considered in the policy updates. The third formulation corresponded to the full dualist approach proposed in this chapter, which exploits both good and bad experiences. These formulations are here referred to as MDGPS, B-MDGPS and D-MDGPS, respectively. All of them were run using the same set of hyper-parameters, except by those related with the dualist constraints. In the case of the planar manipulator 50 iterations of the algorithm were performed and six trajectories were sampled at each iteration from every local policy. In both reaching tasks, the sample with higher return was used to update the good trajectory distribution, while the sample with lowest return was used to update the bad trajectory distribution ($n_g = 1$ and $n_b = 1$). Both dualist trajectory distributions had a fixed covariance. In order to carry out fair comparisons, the noise required by the whole exploration phase was identical for all the three aforementioned cases.

As stated previously, the main motivation to learn from bad experiences is to reduce failures during training. Figure 5.4 shows the accumulated safe-distance cost generated during the exploration phase in the training process of the 3DoF planar manipulator. Observe that the formulation with only bad experiences and our approach iterated more safely during the exploration when compared to the classic MDGPS. Note that B-MDGPS tended to produce fewer failures, in other words, it generated fewer undesired end-effector movements that led to collisions with the obstacle. However, these safer trajectory distributions compromised learning convergence (Figure 5.5), as more iterations were required to train the NN policy in contrast to the other two formulations. On the other hand, the training phase of the proposed D-MDGPS generated less dangerous robot end-effector movement than the standard MDGPS. Moreover, the resulting NN policy showed a performance quite similar to the standard MDGPS. Therefore, the proposed approach offers a good compromise that leads the robot to fail less and smoothly update its policy.

Note that there is a peak in the safe-distance cost at iteration 3 in Figure 5.4. This due to the fact that the robot end-effector is initially quite far away from the target, which can be considered as an undesired behavior based on the given reward function. Consequently, D-MDGPS fits a bad trajectory distribution to these samples, and then by constraint (5.12), the new updated trajectory distributions are different than these low-return trajectories. Despite these updates are quite aggressive and the robot generates trajectories with high safe-distance cost at iteration 3, these executions are now considered bad, and then the updated trajectory distributions do not generate such negative trajectories in the next iteration (iteration 4).

Figure 5.6 shows the accumulated safe-distance cost generated by CENTAURO during the exploration phase of the reaching task. As in the previous experiment, both B-MDGPS and D-MDGPS generated safer trajectories because they involved lower safe distance cost. On the other hand, Figure 5.7 shows the final distance of the right hand with respect to the target. As we can notice, because both B-MDGPS and D-MDGPS force the updated trajectory distribution to be dissimilar to the high safe-distance cost trajectories, the new trajectories generated at the next iteration have lower safe-distance cost. For this reason, we can see strong changes at the following iterations. Unfortunately, because the target is designed to be closer to the cylinder, in comparison to the previous experiment, the proposed

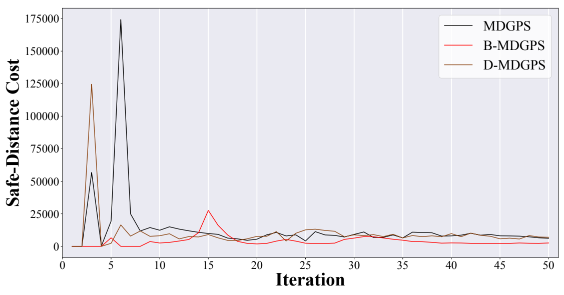


Figure 5.4 *Safe-distance cost with planar robot*. Total safe-distance cost, $\sum_{t=1}^{T} \max(d_{\text{SAFE}} - d(\mathbf{o}_t), 0)$, incurred by the trajectory samples of the planar manipulator robot in each iteration.

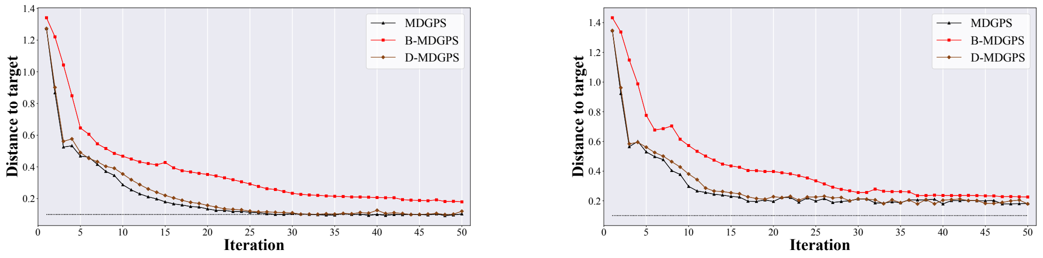


Figure 5.5 *Final distance with planar robot*. Final distance from the end-effector of the planar robot to the target Cartesian pose. Left figure shows the performance on the initial conditions used to train the policy, and the right figure, the performance on the test set.

method tries to maximize the total accumulated reward, and then tries again to generate trajectories closer to the desired Cartesian pose, and then because the stochasticity of the local policies, close again to the obstacle. This situation generated the behavior observed in the Figure 5.7.

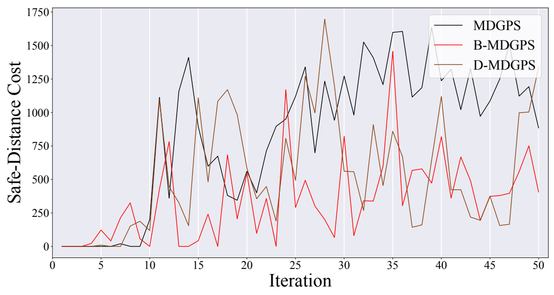


Figure 5.6 Total safe-distance cost, $\sum_{t=1}^{T} \max(d_{\text{SAFE}} - d(\mathbf{o}_t), 0)$, of the trajectory samples generated by the CENTAURO robot in each iteration.

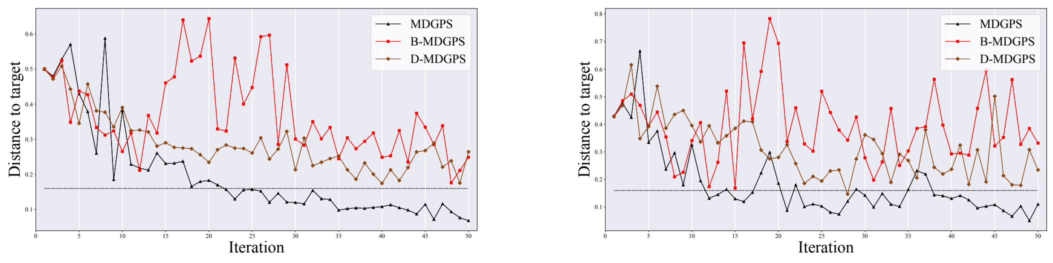


Figure 5.7 Final distance from the CENTAURO's right-hand to the target Cartesian pose. Left figure shows the performance on the initial conditions used to train the policy, and the right figure, the performance on the test set.

5.5 Challenges

A possible limitation of the approach proposed in this chapter is that the optimization problem in (5.6) could be infeasible because there might not be trajectory distributions satisfying all the hard constraints at the same time. Additionally, considering dualist constraints might produce very conservative or aggressive policy updates, then increasing the number of samples required to converge. A possible solution to overcome these problems is to automatically adjust the upper and lower bounds of the good and bad trajectories at each iteration, similarly as done with the bound between the new policy and the TVLG controller approximation of the global policy in MDGPS. A second alternative is to relax the optimization problem and consider the dualist constraints as soft constraints, in such a way the weights of the KL divergence for good and bad trajectory distributions may either be hyperparameters of the algorithm or adjusted by the aforementioned option.

5.6 Summary

In this chapter it is proposed a reinforcement learning algorithm that allows considering both successful and failed executions during the learning process of deep neural network controllers. The method extends the Guided Policy Search algorithm by providing experience from trajectory distributions optimized with dualist constraints. These constraints are aimed at assisting the policy learning so that the trajectory distributions, updated at each iteration, are similar to good trajectory distributions (e.g., sucessful executions) while differing from bad trajectory distributions (e.g. failures). The results of the conducted experiments show that neural network policies guided by trajectories optimized with the proposed method reduce the failures during the policy exploration phase, and therefore encourage safer interactions.

Chapter 6

CONCLUSIONS

Obtaining rational policies is the ultimate goal of robot skill learning. The performance measure assigned to a behavior defines the level of rationality of the control policy regarding a particular task. Because learning implies improving this quantity over time, most robot learning approaches use the interaction data collected during the task execution to directly modify the policy in order to enhance its performance. However, considering the skill acquisition process as an isolated stage, disconnected from a continual learning activity, limits both the models and algorithms that might be used.

This thesis have tried to change the way how robot learning approaches use the interaction data by learning models that are useful for both the current learning process and future ones. In this sense, the robotic motor skills are obtained from alternative objectives and not through the direct optimization of a specific learning criterion. The three *indirect methods* proposed in this thesis seek to be more data efficient and extract more information from the interaction data collected either from expert's demonstrations or the robot's own experience. The main characteristics and benefits of these methods are given in the following section, and potential avenues for future research discussed in 6.2.

6.1 Summary

6.1.1 Learning and transferring shared latent spaces of robotic skills

In chapter 3, it was shown that, in the context of behavioral cloning by imitation learning, the rational behavior captured in the demonstrations can be encoded with a shared latent variable model. By adding a smooth mapping (back-constraint) to this model, a state is projected to the latent space, so-called skill space, from which the action is generated. This state-latent-action mapping is an indirect policy because the action is inferred from the skill variable, compared to a direct policy represented by a state-to-action mapping. That is, the robot policy obtained from the demonstrations is reduced to a conditional distribution that uses only the skill variable.

When several robots should learn the same rational behavior, the skill space obtained for one robot is invariant among the different robots if the state representation of the environment and the influence of an action to the evolution of this state are both independent of the robot executing the task. Therefore, learning the same skill among different robots is formulated as a transfer learning problem, where the state-to-latent map and the skill space of one robot is used to learn the latent representation for others. More specifically, a model pretrained in one robot is transferred to another by fixing the latent variables and the state-to-latent mapping, and optimizing only the latent-to-action mapping for the others. In the chapter, the shared latent space was modeled with a shared GP-LVM, a non-parametric probabilistic model which performs a non-linear dimensionality reduction over the state-action data. The experiments demonstrated that a good imitation performance and a faster learning rate are obtained when the shared latent space for one robot is reused by other robots with similar kinematic structure, compared to processes where learning the shared latent space is carried out independently.

6.1.2 Concurrent discovery of compound and composable policies

In chapter 4, it was shown how learning a rational policy for a complex task with a single-task model-free reinforcement learning (RL) process is reformulated as a two-level hierarchical approach. First, a set of Gaussian policies, constituting the low level of the hierarchy, are composed at the high level by means of state-dependent activation vectors defined for each policy. These activation vectors allow to consider concurrently actions sampled from all the low-level policies and preferences among specific components of the action. Furthermore, two alternatives were proposed to obtain a compound Gaussian policy as a function of the parameters of the low-level policies and their corresponding activation vectors. As a result, the behavior in the complex task is not generated directly by actions sampled from a single policy trained in the task. Instead, the behavior is generated by policies specialized in different aspects of the task.

Additionally, the Hierarchical Intentional Unintentional (HIU) algorithm was proposed in order to learn the different components of the hierarchical policy, by exploiting the off-policy data generated from the compound policy for learning the activation vectors and indirectly the low-level policies concurrently. The resulting multi-task formulation was built on a maximum entropy RL setting to favor exploration during the learning process, and the soft actor-critic (SAC) algorithm used to learn the hierarchical policy. Therefore, the experience collected from the compound policy is exploited in a such a way that not only improves the performance in this task, but also acquire meaningful composable policies in an off-policy manner. Several experiments with composable tasks performed by simulated robots were conducted to validate the proposed approach. The obtained results suggest that HIU-SAC allows to solve the complex task but also to obtain useful composable policies that successfully perform in their respective tasks with comparable or better performances than direct single-task RL formulations.

6.1.3 Exploiting good and bad experiences in policy learning

In chapter 5, it is proposed a model-based trajectory-centric reinforcement learning algorithm that explicitly considers good and bad experiences in order to reduce the occurrence of failures during the learning process. More specifically, the interactions that result in failures are explicitly considered by modeling policies, so-called bad policies, that can reproduce them. Similarly, good policies can be learned by considering only the successful executions. Later, both type of policies are considered in the policy update step by means of dualist constraints, that are KL divergence bounds between the new trajectory distribution and the trajectory distributions induced by the good and bad policies. Thus, the robot's policy is not directly updated from the performance of the resulted behavior. Instead, the policy update includes also the similarity to the good policies and dissimilarity with the bad policies.

In addition, it was proposed to use the trajectories optimized with this method as guiding samples in a guided policy search setting. Specifically, the mirror descent guided policy search (MDGPS) algorithm was used for training a high-dimensional global policy represented by a general-purpose neural network. Thus, neither performance measure nor good-bad policies are considered directly by the global policy, but indirectly through the guiding policies. In this way, it is possible to obtain a deep neural network policy that considers both good and bad behaviors in its learning process, which extends the MDGPS learning capabilities. The approach was evaluated in two reaching tasks using a simulated planar robot and a simulated humanoid robot. The reported results supported the hypothesis that specifying dualist updates reduces the cost related to failures during the exploration phase. Additionally, the results also showed that discarding bad samples from the training dataset in which the global policy is trained on, does not necessarily avoid that the policy generates bad trajectories.

6.2 Open challenges and future work

Learning a latent space from the interaction data have shown benefits in the transfer of robotic skills. However, some challenges need to be addressed to further exploit the potential of learning skill spaces. First, the experiments in this thesis have been conducted with robots with similar kinematic structure, and thus, with equivalent action spaces. For this reason, is an open problem discovering how the skill spaces can be exploited with robots that have significant differences in their actions. However, in view of our obtained results, if the skill space is properly built the proposed method should still lead to good results.

An important motivation for task decomposition is reusing the composable policies in new tasks, however the experiments conducted in this thesis have been limited only in learning the policies but no in their reuse. Future work will be to analyze the behavior of the composable policies when they are reused in different compound tasks. It is important to know how the performance of new compound policies is affected by composable policies obtained in different contexts. Moreover, the experiments carried out in this thesis were focused in tasks that are decomposed in two subtasks. Future work will consider tasks that could be decomposed in more than two subtasks and tasks where the components of the action vector can be assigned completely to specific tasks, e.g. bimanual tasks.

A control policy is sufficient to define either a bad or good behavior, and thus the main motivation for capturing them by policy models. However, trajectory distributions are unimodal, and thus require several of them in order to capture different but still relevant good and bad experiences. Similarly, on-policy data was used in order to update the trajectory distributions, and then, they only captured the recent experience for guiding the neural network policy. Future work will consider alternative and more complex models for capturing both good and bad experiences, and extensions of the algorithm will consider off-policy data. On the other hand, defining bad trajectories in terms of the total reward obtained in one particular task has sense because the performance measure reflex the rationality of the behavior for that task. However, what is unacceptable for that task is not necessarily bad to others. Considering a higher continual learning activity, how these bad policies can influence the learning process of other tasks is an open question.

REFERENCES

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2010.

[2] Marc Toussaint, Helge Ritter, and Oliver Brock. The optimization route to robotics—and alternatives. *KI - Künstliche Intelligenz*, 29(4):379–388, Nov 2015.

[3] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1st edition, 1957.

[4] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific, Belmont, MA, USA, 3rd edition, 2005.

[5] Jan Peters, Daniel D. Lee, Jens Kober, Duy Nguyen-Tuong, J. Andrew Bagnell, and Stefan Schaal. Robot learning. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, chapter 15, pages 357–398. Springer, 2016. 2nd Edition.

[6] Henrik I. Christensen and Gregory D. Hager. Sensing and estimation. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, chapter 5, pages 91–112. Springer, 2016. 2nd Edition.

[7] Christopher G. Atkeson. *Roles of knowledge in motor learning*. PhD thesis, Massachusetts Institute of Technology, 1986.

[8] Eric W. Aboaf, Christopher G. Atkeson, and David J. Reinkensmeyer. Task-level robot learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1309–1310, April 1988.

[9] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[10] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[11] Sergey Levine, Nolan Wagener, and Pieter Abbeel. Learning contact-rich manipulation skills with guided policy search. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 156–163, 2015.

[12] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine learning (ICML)*, 2004.

[13] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2112–2118, 2009.

[14] Petar Kormushev, Sylvain Calinon, and Darwin G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3232–3237, 2010.

[15] Leslie P. Kaelbling and Tomás Lozano-Pérez. Learning composable models of parameterized skills. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 886–893, 2017.

[16] Marc Deisenroth and Carl E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[17] Olivier Sigaud and Freek Stulp. Policy search in continuous action domains: An overview. *Neural Networks*, 113:28–40, 2019.

[18] Carl Henrik Ek. *Shared Gaussian Process Latent Variables Models*. PhD thesis, Oxford Brookes University, 2009.

[19] Katsu Yamane, Yuka Ariki, and Hodgins Jessica. Animating non-humanoid characters with human motion data. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 169–178, 2010.

[20] Aaron P. Shon, Keith Grochow, and Rajesh P.N. Rao. Robotic imitation from human motion capture using gaussian processes. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 129–134, 2005.

[21] Brian Delhaisse, Domingo Esteban, Leonel Rozo, and Darwin Caldwell. Transfer learning of shared latent spaces between robots with similar kinematic structure. In *International Joint Conference on Neural Networks (IJCNN)*, pages 4142–4149, 2017.

[22] Domingo Esteban, Leonel Rozo, and Darwin G. Caldwell. Hierarchical reinforcement learning for concurrent discovery of compound and composable policies. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019. (Accepted), preprint: arXiv:1905.09668.

[23] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Neural Information Processing Systems (NIPS)*, pages 1071–1079, 2014.

[24] Adrià Colomé and Carme Torras. Dual reps: A generalization of relative entropy policy search exploiting bad experiences. *IEEE Transactions on Robotics*, 33(4): 978–985, 2017.

[25] Domingo Esteban, Leonel Rozo, and Darwin G. Caldwell. Learning deep robot controllers by exploiting successful and failed executions. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 1087–1094, 2018.

[26] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[27] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5): 469–483, 2009.

[28] Aude G. Billard, Sylvain Calinon, and Rudiger Dillmann. Learning from humans. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, chapter 74, pages 1995–2014. Springer, 2016. 2nd Edition.

[29] Shin'ichiro Nakaoka, Atsushi Nakazawa, Fumio Kanehiro, Kenji Kaneko, Mitsuharu Morisawa, Hirohisa Hirukawa, and Katsushi Ikeuchi. Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances. *The International Journal of Robotics Research*, 26(8):829–844, 2007.

[30] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143:1–143:14, 2018.

[31] Leonel Rozo, Sylvain Calinon, Darwin G. Caldwell, Pablo Jiménez, and Carme Torras. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 32(3):513–527, June 2016. ISSN 1552-3098.

[32] Eric L. Sauser, Brenna D. Argall, Giorgio Metta, and Aude G. Billard. Iterative learning of grasp adaptation through human corrections. *Robotics and Autonomous Systems*, 60(1):55–71, 2012.

[33] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29 (13):1608–1639, 2010.

[34] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5628–5635, 2018.

[35] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Neural Information Processing Systems (NIPS)*, pages 1087–1098, 2017.

[36] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning (CoRL)*, pages 357–368, 2017.

[37] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[38] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical Report 166, Cambridge University Engineering Departtment, 1994.

[39] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[40] Roland Hafner and Martin Riedmiller. Reinforcement learning on an omnidirectional mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 418–423, 2003.

[41] Jun Morimoto and Kenji Doya. Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, 36(1):37–51, 2001.

[42] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1–2):1–142, 2013.

[43] Lucian Buçoniu, Tim de Bruin, Domagoj Tolić, Jens Kober, and Ivana Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28, 2018.

[44] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[45] Voot Tangkaratt, Abbas Abdolmaleki, and Masashi Sugiyama. Guide actor-critic for continuous control. In *International Conference on Learning Representations (ICLR)*, 2018.

[46] Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86 (2):153–173, May 2017.

[47] Duy Nguyen-Tuong and Jan Peters. Model learning for robot control: a survey. *Cognitive Processing*, 12(4):319–340, 2011.

[48] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018.

[49] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519, 2016.

[50] Axel Rottmann and Wolfram Burgard. Adaptive autonomous control using online value iteration with gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2106–2111, 2009.

[51] Michael C. Yip and David B. Camarillo. Model-less feedback control of continuum manipulators in constrained environments. *IEEE Transactions on Robotics*, 30(4): 880–889, 2014.

[52] Igor Mordatch, Nikhil Mishra, Clemens Eppner, and Pieter Abbeel. Combining model-based policy search with online model learning for control of physical humanoids. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 242–248, 2016.

[53] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.

[54] Duy Nguyen-Tuong, Jan Peters, and Matthias Seeger. Local gaussian process regression for real time online model learning and control. In *Neural Information Processing Systems (NIPS)*, pages 1193–1200, 2008.

[55] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[56] Aaron P. Shon, Keith Grochow, Aaron. Hertzmann, and Rajesh P.N. Rao. Learning shared latent structure for image synthesis and robotic imitation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 129–134, 2006.

[57] Fanny Ficuciello, Pietro Falco, and Sylvain Calinon. A brief survey on the role of dimensionality reduction in manipulation learning and control. *IEEE Robotics and Automation Letters*, 3(3):2608–2615, 2018.

[58] Adrià Colomé and Carme Torras. Dimensionality reduction and motion coordination in learning trajectories with dynamic movement primitives. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1414–1420, 2014.

[59] William Curran, Tim Brys, David Aha, Matthew Taylor, and William D Smart. Dimensionality reduced reinforcement learning for assistive robots. In *AAAI Fall Symposium Series*, 2016.

[60] Sebastian Bitzer, Matthew Howard, and Sethu Vijayakumar. Using dimensionality reduction to exploit constraints in reinforcement learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3219–3225, 2010.

[61] Matthew Field, David Stirling, Zengxi Pan, and Fazel Naghdy. Learning trajectories for robot programing by demonstration using a coordinated mixture of factor analyzers. *IEEE Transactions on Cybernetics*, 46(3):706–717, 2016.

[62] Sinno Jialin Pan, James T. Kwok, and Qiang Yang. Transfer learning via dimensionality reduction. In *AAAI Conference on Artificial Intelligence*, pages 677–682, 2008.

[63] Botond Bócsi, Lehel Csató, and Jan Peters. Alignment-based transfer learning for robot models. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2013.

[64] Ndivhuwo Makondo, Benjamin Rosman, and Osamu Hasegawa. Knowledge transfer for learning robot models via local procrustes analysis. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 1075–1082, 2015.

[65] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[66] Markus Schneider and Wolfgang Ertel. Robot learning by demonstration with local gaussian process regression. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 255–260, 2010.

[67] Mauricio A. Álvarez, Lorenzo Rosasco, and Neil D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends in Machine Learning*, 4(3): 195–266, 2012.

[68] Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems (NIPS)*, pages 329–336, 2004.

[69] Neil D. Lawrence and Joaquin Quiñonero-Candela. Local distance preservation in the gp-lvm through back constraints. In *International Conference on Machine Learning (ICML)*, pages 513–520, 2006.

[70] Nikos G. Tsagarakis, Darwin G. Caldwell, and et al. Walk-man: a high performance humanoid platform for realistic environments. *Journal of Field Robotics*, 2016.

[71] Nikos G. Tsagarakis, Stephen Morfey, Gustavo Medrano Cerda, Li Zhibin, and Darwin G. Caldwell. Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 673–678, 2013.

[72] L. Baccelliere, N. Kashiri, L. Muratore, A. Laurenzi, M. Kamedula, A. Margan, S. Cordasco, J. Malzahn, and N. G. Tsagarakis. Development of a human size and strength compliant bi-manual platform for realistic heavy manipulation tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[73] GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, since 2012.

[74] 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 *International Conference on Learning Representation (ICLR)*, 2016.

[75] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[76] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17(93):1–50, 2016.

[77] Katharina Mülling, Jens Kober, and Jan Peters. Learning table tennis with a mixture of motor primitives. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 411–416, 2010.

[78] Takayuki Osa, Jan Peters, and Gerhard Neumann. Hierarchical reinforcement learning of multiple grasping strategies with human instructions. *Advanced Robotics*, 32(18): 955–968, 2018.

[79] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6244–6251, 2018.

[80] Nathan Sprague and Dana Ballard. Multiple-goal reinforcement learning with modular sarsa(o). In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1445–1447, 2003.

[81] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[82] Jens Kober and Jan Peters. Learning elementary movements jointly with a higher level task. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 338–343, 2011.

[83] Chunming Liu, Xin Xu, and Dewen Hu. Multiobjective reinforcement learning: A comprehensive overview. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(3):385–398, March 2015.

[84] Christopher Simpkins and Charles Isbell. Composable modular reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2019.

[85] Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural Comput.*, 14(6):1347–1369, June 2002.

[86] Eiji Uchibe and Doya Doya. Combining learned controllers to achieve new goals based on linearly solvable mdps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5252–5259, 2014.

[87] Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 761–768, 2011.

[88] Serkan Cabi, Sergio Gómez Colmenarejo, Matthew W Hoffman, Misha Denil, Ziyu Wang, and Nando De Freitas. The intentional unintentional agent: Learning to solve many continuous control tasks simultaneously. In *Conference on Robot Learning (CoRL)*, pages 207–216, 2017.

[89] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning (ICML)*, pages 1312–1320, 2015.

[90] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representation (ICLR)*, 2016.

[91] Zhaoyang Yang, Kathryn Merrick, Hussein Abbass, and Lianwen Jin. Hierarchical deep reinforcement learning for continuous action control. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5174–5184, Nov 2018.

[92] Martin Riedmiller, Roland Hafner, Thomas Lampe, Michael Neunert, Jonas Degrave, Tom Van de Wiele, Volodymyr Mnih, Nicolas Heess, and Tobias Springenberg. Learning by playing - solving sparse reward tasks from scratch. In *International Conference on Machine Learning (ICML)*, volume 80, pages 4344–4353, 2018.

[93] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, Carnegie Mellon University, 2010.

[94] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

[95] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, pages 1861–1870, 2018.

[96] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[97] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[98] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2018.

[99] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[100] William Montgomery and Sergey Levine. Guided policy search as approximate mirror descent. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[101] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[102] Emanuel Todorov and Weiwei Li. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference*, pages 300–306, 2005.

[103] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4906–4913, 2012.

[104] Rudolf Lioutikov, Alexandros Paraschos, Jan Peters, and Gerhard Neumann. Sample-based informationl-theoretic stochastic optimal control. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3896–3902, 2014.

[105] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3381–3388, 2017.

[106] Daniel H. Grollman and Aude G. Billard. Robot learning from failed demonstrations. *International Journal of Social Robotics*, 4(4):331–342, 2012.

[107] Akshara Rai, Guillaume De Chambrier, and Aude Billard. Learning from failed demonstrations in unreliable systems. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, pages 410–416, 2013.

[108] Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. Inverse reinforcement learning from failure. In *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pages 1060–1068, 2016.

[109] Sungjoon Choi, Kyungjae Lee, and Songhwai Oh. Robust learning from demonstration using leveraged gaussian processes and sparse-constrained optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 470–475, 2016.

[110] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *AAAI Conference on Artificial Intelligence*, pages 1607–1612, 2010.

[111] David H. Jacobson and David Q. Mayne. *Differential dynamic programming*. American Elsevier Pub. Co New York, 1970.

[112] Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems (RSS)*, 2014.

[113] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 528–535, 2016.