Robotic skill learning

A literature study on how a novice robot becomes an expert

by

R.E.S. Maesen

Author:	R.E.S. Maessen
Student number:	4564200
Supervisor:	Dr. L. Peternel
Institution:	Delft University of Technology
Study:	Msc. Robotics
Date:	Monday 14th February, 2022

T̃U Delft

Summary

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent

per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Contents

List of Abbreviations

L2S learning to search.

AC actor-critic.

BC Behavioural Cloning.

COACH Corrective Advice Communicated by Humans.

DMP Dynamic Movement Primitive.
DoF Degrees of Freedom.
DP Dynamic Programming.
DS Dynamic System.

EM Expectation Maximization.
EMG electromyography.

GHMM Growing Hidden Markov Model.
GMM Gaussian Mixture Model.
GMR Gaussian Mixture Regression.
GP Gaussian Process.
GPR Gaussian Process Regression.

HBM Hierarchical Bayesian Model.
HHMM Hierarchy Hidden Markov Model.
HMM Hidden Markov Model.
HSMM Hidden Semi-Markov Model.

i.i.d. independent and identically distributed.
I2RL incremental IRL Expert.
IL Imitation Learning.
ILWR incremental LWR.
Inf.Th. information-theoretic insights.
IRL Inverse Reinforcement Learning.
IRLF Inverse Reinforcement Learning from Failure.

LGP Local GP.
LOLS Locally Optimal Learning to Search.
LWPR Locally Weighted Projection Regression.
LWR Locally Weighted Regression.

MC Monte Carlo.

MDP Markov Decision Process.

MMP maximum margin planning.

MP Motion Primitive.

MPC model predictive control.

PG policy gradient.

PiH peg-in-hole.

ProMP Probabilistic Movement Primitive.

PS policy search.

RBF Radial Basis Function.

RL Reinforcement Learning.

SOGP Sparse Online Gaussian Processes.

SPD symmetric positive definite.

TD Temporal-Difference.

TP task-parametrized.

TP-GMM TP GMM.

TP-HSMM TP HSMM.

List of Figures

List of Tables

1

Introduction

The last few years robotic learning, to learn a robot a certain skill, has increased popularity. Traditionally, robots skills were directly programmed. This code, controlling the behaviour of the robot, would need to be created such that the robot is prepared for all possible scenarios. This usually meant that the code would exist of hundreds or even thousands of parts, each consisting of a certain type of action [1]. While for some shielded environments this might still be useful, robots nowadays often have to execute alongside humans. This demands for a high safety level and the ability of robots to adapt to uncertainties in the environment. The latter is especially relevant for environments with humans, as humans are unpredictable.

There thus is a need for robots who can adapt to their environment. In other words, there is need for robots who can learn. In this report, robot learning will be examined. This will be done by trying to answer the following research question:

> How do methods, found in literature, enable continuous skill learning of robots, and how does the learned skill change during the learning process?

To narrow the scope of this research, one important choice was made. Robotic learning can either be on- or offline. In this research the focus will only be on the online learning. Before explaining why, a description of this term should be given as diferent literature often uses different descriptions. Online learning is seen in this report as any method which allows for some change in the model (dictating the behaviour of the robot), throughout the learning process. This can both mean that the robot executes a behaviour during the learning stage where it is seen that the behaviour in iteration 1 is different from that in iteration 2. And also, the updating of a model which cannot directly be seen in the real world.

To elaborate why only online learning will be examined, first offline learning should be explained. In offline learning, data can be acquired based on which a model, representing the robotic skill, is trained. This model is then used in an autonomous stage to let the robot execute certain tasks. Advantages of this method is that the operator can select the data or adjust it, to achieve the best learning performance [2]. A downside to pure offline learning, is that no feedback about the behaviour is provided. This can partly be addressed by providing a variety of data (e.g. different tasks or different starting points) or by correcting the skill during the autonomous stage. But this again requires a high level of robot intelligence to recognize the human device or advise a strategy to correct its performance on sensorimotor level. Especially for complex tasks, this can be time-consuming and sometimes even unsuccessful. Therefore, online learning might be more appropriate. This method assumes that the robotic skill is gradually formed during the learning stage [2]. An advantage of this approach is that the transition between the learning and execution stage can be direct and autonomous. This approach also allows for useful feedback about the performance while the robotic skill is constructed. Another motivation for online model learning is that it is not possible to cover the complete state space with data beforehand, but only the interesting state space regions are known during execution [3]. Therefore, online learning will require incremental acquisition of knowledge and, possibly, even partial forgetting of the recorded information in order to cope with errors as well as change.

To answer the research question, two topics will be discussed in this paper. Each of these topics relates to a part of the research question. The first topic is the enabling of continuous skill learning of robots, and the second the skill change during the learning process. These will, respectively, be discussed in chapters 2 and 3. The main findings will be discussed in chapter 4, and in chapter 5 a conclusion, answering the research question, will be given.

2

Skill Learning Methods

An important goal of robotics is to allow robots to perform certain tasks in a smooth, natural way. In the past, this behaviour could be programmed using a direct approach, also known as a *"hard coded"* approach. While this approach is useful for well-structured environments, problems arise when this is not the case. Directly programmed robots are, for example, unable to adapt to changing environment. This is especially important when these robots work collaboratively with a human, as safety becomes a top priority. But even when this is not the case, adapting to environment prevent the robot of harming itself and/or the environment. Two learning approaches have proved to be successful in these case: Imitation Learning (IL) and Reinforcement Learning (RL). The former uses demonstration, exerted by an expert, to train the robot. The latter takes a trial-and-error approach, where the environment is explored in order to perform a desired task as good as possible. An overview of advantages and disadvantages of these three approaches is shown in table 2.1.

Table 2.1: Comparison of three method based on which robot can require skills [4]. The first method, *direct programming*, is "hard coded", whereas the other two, *IL* and *RL*, allow for continuous skill learning.

Method	Advantages	Disadvantages
Direct Programming	• Complete control of movement robot to the lowest level	• Time-consuming • Error-prone • Not scalable • Not reusable
IL	• No manual program of behaviour • Solving problems which can be demonstrated, but not succinctly specified in mathematical form • Low number of demonstrations needed • Save design • High reliability	• Correspondence problem: demonstrators actuation differs from human • Cannot deal properly with suboptimal data • Covariant shift/compounding error when dealing with other input distributions • Performance robot is limited by skill of expert
RL	• No manual program of behaviour • Robot can learn tasks that humans cannot demonstrate • Novel ways to reach a goal can be discovered	• Requires lots of trials • No control over the actions of the robot (unsafe) • Robot has only indirect information about the goal • Need to specify reward function, policy parameterization, exploration magnitude/strategy and initial policy

A recent paper by Kroemer, Niekum, and Konidaris [5], reviewed robotic learning for manipulation. In this chapter the focus will be on the aspect of learning a policy. More specific, the focus will only be on learning a policy directly. Another approach for robots to learning is by skill transfer. This enables the robot to transfer a skill, learned in one task, to another task. An advantage of this method is that it can increase the efficiency of learning [5]. However, as this specific topic does not allow detecting how the skill of a robot has been learned and has been improved thought the learning process, it was decided that it would be excluded for further research.

This chapter is dedicated to present the theoretical background behind the two different learning approaches which are able to adapt to the environment: IL and RL. To start, in section 2.1 the Markov Decision Process (MDP) is explained, which is often used in robot teaching. This is followed by an overview of the IL approach in section 2.2 and the RL approach in section 2.3. For both approaches, this includes both a problem definition and an explanation of some methods. In section 2.5, an overview of the different robot teaching approaches is given. Lastly, a summary of the chapter is presented in section 2.6.

2.1. Markov Decision Process

Skill learning often requires the learning of a policy which best represents the desired behaviour of a robot. To learn this policy, different methods could be employed. Most of these methods use the same type of problem. Looking at skill learning, often MDP is used as underlying decision-making [5, 6]. MDP is described as a tuple $\langle S, A, R, T, \gamma \rangle$, where:

- $S \subseteq \mathbb{R}^n$ is a finite set of N states
- $A = \{a_1, \ldots, a_k\} \subseteq \mathbb{R}^m$ is a set of k actions
- $R : \mathcal{S} \mapsto \mathbb{R}$ is the reward function expressing the immediate reward of executing an action a in state s and transitioning into state s', this is bounded by absolute value $R_{\max}$ [7]
- $T = P(s \mid a, s')$ is the transition function, giving the probability distribution that states s' are reached after executing action a in state s.
- $\gamma \in [1, 0)$ a discount factor expressing the agent's preference for immediate over future rewards.

To solve these MDPs, a policy $\pi : S \times A \to [0, 1]$ should be found, that maximizes the expected sum of rewards for a specific problem [6]. In order for the MDP to hold, it should satisfy the Markov property. This property states, that the next state s' and the reward only depend on the previous state s and action a, and not on any additional information about the past states or action.

2.2. Imitation Learning

The process of learning a skill using IL, consist of three steps: observing an action, representing the action and reproducing the action [8]. For the first step, a demonstration of the desired task, performed by an expert, is required. There are different types of demonstration methods which have been used in literature [8–10]:

- *Kinaesthetic Teaching*: where the human holds the robot limb and physically guides it to perform the desired task.
- *Teleoperation*: where the human performs and teaches the task by controlling the robot through human-robot interfaces.
- *Passive Observation*: where the human performs the task on his/her own, while the robot then learns from the observation.
- *Shadowing*: where the executed behaviour of the human is recorded using sensors and the robot tries to match or mimic the teacher's motions while executing the task.

The remaining of this section consist of a general problem definition of the IL approaches in section 2.2.1. Followed by two broadly used IL approach: Behavioural Cloning (BC) and Inverse Reinforcement Learning (IRL), discussed in sections 2.2.2 and 2.2.3, respectively. This also includes some methods which can be used for online learning.

2.2.1. Problem definition Imitation Learning

There are numerous methods to tackle IL. These try to solve the same problem. Starting, the goal of IL is to learn a policy π_θ which can reproduce the behaviour demonstrated by the experts for performing a certain task [11]. The behaviour of the expert (and eventually the learner) can be described as a trajectory $\tau = [(s_1, a_1), \ldots, (s_n, a_n)]$, which consist of a sequence of state-action pairs. This can also be represented using the feature ϕ. To learn a policy, it is assumed that an expert is exerting a certain policy π^E. As this cannot directly be transfer to the robot, it is instead done by demonstrations. The demonstrations consist are represented by a dataset $\mathcal{D} = (\tau_i, r_i)_{i=1}^N$, where the reward signal r is optional. Using this dataset, the IL problem can be reframed as an optimization problem:

$$\pi^* = \arg\min\left(D\left(q\left(\phi\right), p\left(\phi\right)\right)\right), \tag{2.1}$$

where the optimal policy π^* is learned, such that the difference between the learned sequence and the demonstrated trajectory, is minimized. Here, $q(\phi)$ is the distribution of features induced by the experts' policy, $p(\phi)$ the feature distribution induced by the learner, and $D(q, p)$ is the similarity measure between q and p.

Learning approaches

There are different IL approaches which can be used to find the policy. Often these approaches are categorized into two groups: BC and IRL. BC obtains the policies using supervised learning [12]. It does this by directly mapping the state. An advantage of this technique is that it is quite simple. However, it only tends to work successfully with large amounts of data, due to compounding error caused by covariate shift. IRL describes the policy as the solution of an optimization or planning problem [11]. As this method learns a cost function which prioritizes the entire trajectory over others, compounding errors is not an issue [12]. Given a reward signal, IRL can obtain the policy such that it maximizes the expected return $J(\pi)$. As the reward function is unknown, it needs to be recovered from the expert demonstrations (here we assume that the behaviour exerted in the demonstrations is (approximately) optimal). By maximizing the expectation of the accumulated rewards $J(\hat\pi)$, the policy can then be found. An overview of the advantages and disadvantages of both approaches is given in table 2.2.

Policy Representation

As it is the eventual goal of imitation learning to define a policy which best represents the behaviour of the expert, an important consideration is how the policy should be represented. This should be chosen such that it can capture the desired behaviour properly [11]. Besides, the complexity of the policy also should be taken into account when choosing a representation; If the complexity of the representation is increased, the model will probably capture the desired behaviour better, however it will in most cases also lead to an increasing of the learning time, and it would ~~also~~ require more training data.

There are three types of policy representation available [11]: task-level abstraction, trajectory-level abstraction, and action-state space abstraction. A summary of these methods is shown in table 2.3. In

Table 2.2: Advantages and disadvantage of the two main IL approaches based on multiple papers [12–15].

	Advantages	Disadvantages
BC	• Computational efficient • Performance in high-dimensional space	• Compounding error caused by covariant shift • Often large amount of data required for stable policy • Performance in dynamical and suboptimal space
IRL	• No compounding error • Good performance in suboptimal space	• Expensive to run • Can be impractical in real-world • Performance in high-dimensional space • Matches at high level outcomes instead of actions

task-level abstraction, the learner learns a policy that generates an option $o \in \mathcal{O}$, where $\mathcal{O}$ is the set of options each consisting of a set of actions or trajectories. Here, the policy maps a given state x_t and context s to a sequence of options (T is the horizon of the task). As complex task are often difficult to model as a single motion, BC methods can model them as a sequence of movement primitives. In trajectory-level abstraction, a policy maps a context s to a trajectory τ, which is a sequence of the state of the system x (and control inputs u). In action-state space level abstraction, the policy maps states of the system x_t and context s to control inputs u_t as. Each of these abstraction methods have been applied in robotics. However, the most frequently used for robotic manipulator is the trajectory-based [11].
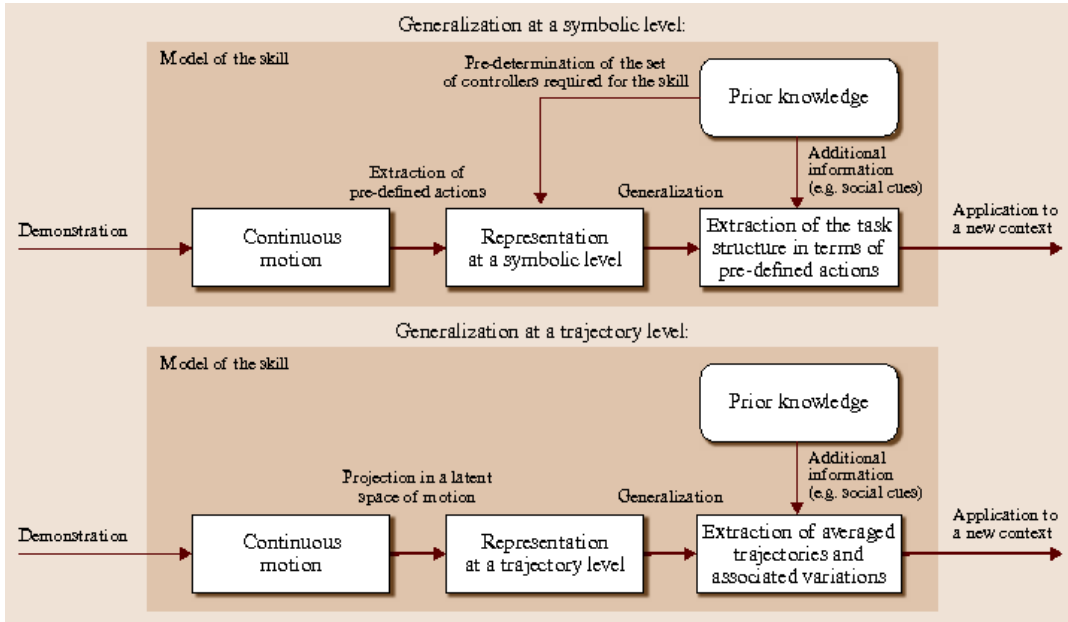
Figure 2.1: Illustration of the different levels of representation for describing the skill [16]. The upper figure shows the model of the skill for a symbolic level, whereas the lower illustrated the trajectory level.

2.2.2. Behavioural Cloning Methods

Behavioural Cloning (BC) is the simplest form of IL. This method tries to learn the policy, by directly mapping from state/context to trajectories/actions or to the control input [11]. An overview of the BC approach is given in algorithm 1. The first step is to obtain the experts' demonstrations. Different methods for doing so were discussed earlier. Next, an appropriate policy abstraction has to be chosen. Options for this were given in table 2.3. In addition, an objective function $\mathcal{L}$, which represents the similarity between the demonstrated behaviours and the learner's policy, should be chosen. Some well known and frequently used loss functions are: quadratic loss function (also known as ℓ_2-loss function), ℓ_1-loss function (also known as absolute loss function), log-loss function, hinge loss and the Kullback-Leibler Divergence. Now, using the dataset of demonstration, the policy parameters θ can be learned.

BC problems can be formulated a supervised learning problem [11]. For such problems, the policy is obtained using a regression problem. However, it should in all cases be ensured that the trajectory is physically feasible, and therefore naive regression method are not always the best choice. To ensure

---

Algorithm 1 Abstract of behavioural cloning

1: Collect a set of trajectories demonstrated by the expert $\mathcal{D}$
2: Select a policy representation π_θ
3: Select an objective function $\mathcal{L}$
4: Optimize $\mathcal{L}$ w.r.t. the policy parameter θ using D
5: **return** optimized policy parameters θ

---

the feasibility, constraints should be added. In some cases, it is possible that the constraints implicitly satisfy using a regression method. However, in most cases it is more convenient to use a policy which explicitly satisfies some constraints. To learn these policies, the regression methods are used in a way that the constraints satisfies.

~~Generally, the BC method can be divided into Dynamic System (DS) based and statistically/probability based approaches [17]. The former method generally tries to encode the dynamic attractor landscape in the state space (i.e. typically position and velocity) of the demonstration [17]. A big advantage of this method is that it aims to be robust to dynamical changes in the environment [16]. The latter approach uses probability theory to encode the spatial or temporal pattern using joint probability density [17]. This is done by using mathematical models to describe and infer the models [18].~~

There are different method which apply the principle of BC for learning of the policy. In the next part of this section some methods, frequently used in literature, are explained. It should be noted that most of these methods require some adaptation of the original method, in order to be applicable for online learning.

Dynamic Movement Primitives

Motion Primitives (MPs) are often used to represent and learn basic movements in robotics. A frequently method for this representation is Dynamic Movement Primitive (DMP), which was originally introduced by by Ijspeert et al. [19, 20]. DMP provides a framework for the motor representation, based on a nonlinear dynamic system [21]. In recent research, DMP has, among other things, been used for skill learning of clothing assistance tasks [22], skill learning of a co-manipulation tasks [23], and skill learning via a wearable device using DMP [21].

There are two general types of DMPs: discrete and rhythmic [24]. Discrete DMPs are used to encode a point-to-point motion into a stable dynamical system. Whereas, rhythmic DMPs are used to encode motion followed from a rhythmic motion. DMPs are defined using two sets of equations: the transformation system and the canonical system. The transformation system, can be written as a first-order notation of damper spring model [20]

$$\tau \dot{z} = \alpha_z \left(\beta_z (g - y) - z \right) + f(x),$$
$$\tau \dot{y} = z,$$

(2.2)

where y is the state/position of the system, $\dot{y}$ the velocity of the joint trajectory, $\ddot{y}$ the acceleration, τ a time constant, g the goal state, α_z and β_z the gain terms representing the stiffness and damping, respectively, and f forced item used to restrain the repair of the trajectory. Looking at the forcing term $f(x)$, if this is equal to 0, the transformation equations represent a globally stable second-order linear system with $(z, y) = (0, g)$ as a unique point attractor [20]. An example of this has been given in fig. 2.2. To allow more complex trajectories to the goal, f can be defined as a time-dependent, nonlinear function. This results into a nonlinear DS which can be difficult to solve. Therefore, an additional differential equation, called the *canonical system*, is introduced. For *discrete DMPs*, this system can be written as:

$$\tau \dot{x} = \alpha_x x,$$

(2.3)

where α_x is a predefined constant and x the phase variable. Defining $f(x)$ as a linear combination of N nonlinear Radial Basis Functions (RBFs), will ensure that the robot will follow any smooth trajectory from an initial position y_0 to the final configuration g [24]
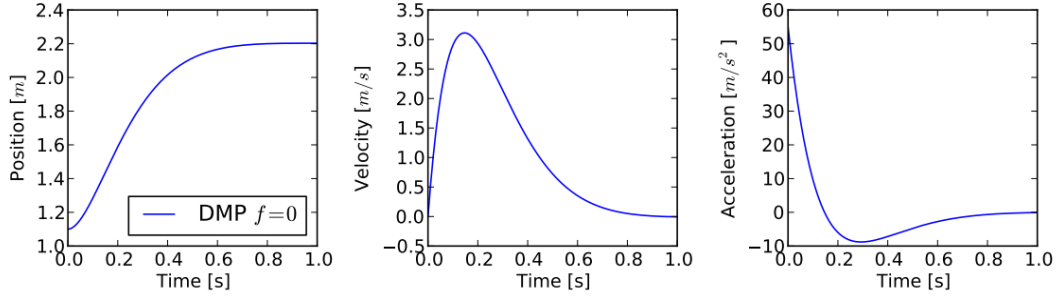
Figure 2.2: Position, velocity and acceleration of an executed DMP without any external force ($f = 0$) [25].

$$f(x) = \frac{\sum_{i=1}^{N} w_i \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x, \tag{2.4}$$

where the basis function is defined as $\Psi_i(x) = \exp(-h_i(x - c_i)^2)$, c_i are the centres of the function distribution along the phase of the movement and h_i their widths. The parameter w, can here be adjusted using learning algorithms, in order to produce complex trajectories.

Rhythmic DMPs, use a transformation system which is quite similar to its discrete variant (eq. (2.2)). Instead of the gain α_z, a frequency Ω is used. And the forcing item is instead dependent on a phase angel of the oscillator in polar coordinates ϕ: $f(\phi)$. The canonical system is then given as:

$$\tau\dot{\phi} = 1, \tag{2.5}$$

Similar to the discrete variant in eq. (2.4), $f(\phi)$ is defined with N kernels

$$f(\phi) = \frac{\sum_{i=1}^{N} \Psi_i(\phi) w_i r}{\sum_{i=1}^{N} \Psi_i(\phi)}, \tag{2.6}$$

where $\Psi_i(\phi) = \exp(h(\cos(\phi - c_i) - 1))$, the weights are uniformly distributed along the phase's pace, and r is used to modulate the amplitude of the periodic signal.

To learn the encoded trajectories, a regression method should be used. One method which is frequently used for online learning problems is Locally Weighted Regression (LWR) [24]. This method is suitable for these types of problems, as it update the existing models fast [23]. In essence, the goal is to find the appropriate weight w_i of f. To formulate the function approximation problem, the target function f_{target} should be defined. This can be done by rewriting eq. (2.2), resulting in

$$f_{target} = \tau^2 \ddot{y}_{demo} - \alpha_z \left(\beta_z \left(g - y_{demo}\right) - \tau\dot{y}_{demo}\right). \tag{2.7}$$

The weight of f can than be determined by minimizing the error criterion $J_i = \sum_{t=1}^{P} \phi_i(t)(f_{target}(t) - w_i\xi(t))^2$, with $\xi(t) = x(t)(g - y_0)$ for a discrete system and $\xi(t) = r$ for a rhythmic system [20]. A graphical example of this has been given in fig. 2.3.

Of both the discrete and rhythmic DMP approach, some extensions exist. Starting with discrete DMPs, the classical approach is meant for single Degrees of Freedom (DoF) motions. In order to obtain multidimensional motions, the transformation system eq. (2.2) have to be repeated while the canonical system eq. (2.3) is shared. This is not a problem when the evolution of the different DoFs are independent. However, a problem arises for orientation representation, where the elements are constraint. Extensions which could be applied in these types of situation are Quaternion MDP[26], rotation matrix MDP [27], and symmetric positive definite (SPD) matrices [28].

Figure 2.3: Example of the learning of DMP using LWR, where: (left) the target function f_{target} (blue) and the predicted function $f_{predicted}$ (dashed green), (middle) 8 Gaussian functions meaning 8 weights ($|w| = 8$), and (right) the local linear model [25].

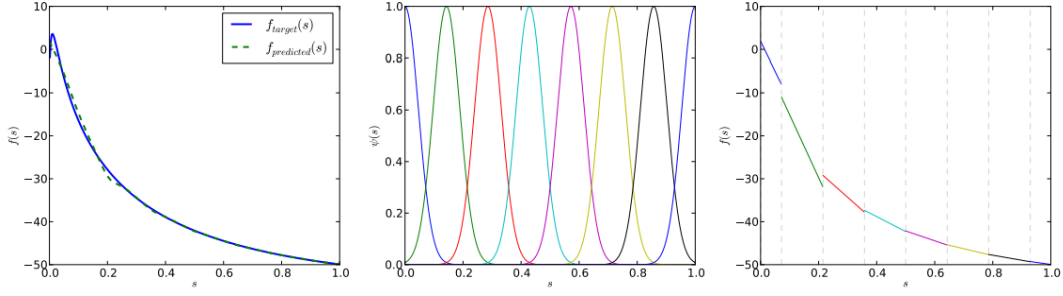Typically, DMP is not used for online learning. A reason for this is the usage of only one demonstration. In [29], learning scheme was presented which iteratively updated the DMP. Here, the data obtained by exerting the learned motion, was used as new input to learn the DMP.

Probabilistic Movement Primitives
A disadvantage of DMP is that it can only represent the mean solution, which is known to be suboptimal [30]. Instead of using such a deterministic approach, a probabilistic approach called Probabilistic Movement Primitive (ProMP) has be introduced by Parachos et al. [31]. This allows for the direct encoding of an optimal behaviour for a system with linear dynamics, quadratic costs and a Gaussian noise. This method provides a single framework for all the desirable properties of a MP: co-activation, modulation, optimality, coupling, learning, temporal scaling, and rhythmic movements [31].

ProMPs uses multiple trajectories to define a single movement, which automatically results into a distribution over the trajectories. This different from DMP, which only uses one trajectory. A single trajectory $\tau = \{q_t\}_{t=0...T}$ can be compactly represented by a weight vector w. Using this will reduce the number of model parameters. The state vector y_t for a single time step, can now be defined

$$y_t = \left[\begin{array}{c} q_t \\ \dot{q}_t \end{array} \right] = \mathbf{\Phi}_t^T w + \epsilon_y, \tag{2.8}$$

where $\mathbf{\Phi}_t = [\phi_t, \dot{\phi}_t]$ describes a $n \times 2$ time-dependent basis matrix for the joint position q_t and velocities $\dot{q}_t$, n defines the number of basis functions, and $\epsilon_y \sim \mathcal{N}(0, \Sigma_y)$ is a zero-main independent and identically distributed (i.i.d.) Gaussian noise. The probability of observing the whole trajectory $p(\tau \mid w)$ is then defined as

$$p(\tau \mid w) = \prod_{t=1}^{T} \mathcal{N}\left(y_t \mid \Phi_t w, \Sigma_y\right). \tag{2.9}$$

As the expectation is that different demonstrations of the same movements, will be slightly different, there are different weight vectors w_n needed to represent the n different instances of a movement [32]. To capture this variance in the trajectories, the distribution $p(w; \theta)$ over the weight vector w, with parameter θ, is introduced. In most cases, this distribution will be Gaussian, where the parameter vector $\theta\{\mu_w, \Sigma_w\}$ specifies the mean μ_w and variance Σ_w of w [30]. Besides using a Gaussian distribution, more complex distributions like a Gaussian Mixture Model (GMM) could also be used for this. By marginalizing out the weights w, the trajectory distribution $p(\tau; w)$ can now be computed

$$p(\tau; \theta) = \int p(\tau \mid w) p(w; \theta) dw. \tag{2.10}$$

This distribution $p(\tau; w)$ defines a Hierarchical Bayesian Model (HBM), whose parameters are given by the observation noise variance Σ_y and the parameters θ of $p(w; \theta)$.

The choice of basis function depends on the type of movement. The movement can either be rhythmic or stroke-based [31]. For stroke-based movements, a Gaussian basis function is used, while for rhythmic movements, a Von-Mises basis functions is used to model periodicity in the phase variable. These basis functions are in most cases normalized, resulting in the normalized basis function ϕ_t.

Up to this point, it was considered that each DoF was modelled independently. However, often the movement of the multiple joints have to be coordinated. A method for doing this, is to introduce the phase variable z_t, which couples the mean of the trajectory distribution [31]. As it is often also desired to encode higher-order moments of the coupling, the model is extended to multiple dimensions. Here, each dimension maintains a parameter vector w_i. They can be combined into a weight vector $w = [w_1^T, \ldots, w_n^T]$. The basis matrix Φ_t extends now to a block-diagonal matrix, which contains the basis functions and their derivatives for each dimension. The observation vector y_t consist of the angles and velocities of all joints. The probability of an observation y_t at time t is given by

$$p\left(\boldsymbol{y}_t \mid \boldsymbol{w}\right) = \mathcal{N}\left(\left[\begin{array}{c} \boldsymbol{y}_{1,t} \\ \vdots \\ \boldsymbol{y}_{d,t} \end{array}\right] \mid \left[\begin{array}{ccc} \boldsymbol{\Phi}_t^T & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \boldsymbol{\Phi}_t^T \end{array}\right] \boldsymbol{w}, \boldsymbol{\Sigma}_y\right) = \mathcal{N}\left(\boldsymbol{y}_t \mid \boldsymbol{\Psi}_t \boldsymbol{w}, \boldsymbol{\Sigma}_y\right) \tag{2.11}$$

For MP representations, it is important that the parameters of a single primitive are easy to acquire from demonstration [31]. To facilitate the estimations of these parameters, a Gaussian distribution for $p(w; \theta) = \mathcal{N}(w \mid \mu_w, \Sigma_w)$ over the parameters w is assumed. This could, however, also be done using other types of probability distribution, e.g. a mixture model. The distribution of the state $p(y_t \mid \theta)$ for time step t is then given by

$$p\left(y_t; \theta\right) = \int \mathcal{N}\left(y_t \mid \boldsymbol{\Psi}_t^T \boldsymbol{w}, \Sigma_y\right) \mathcal{N}\left(\boldsymbol{w} \mid \mu_{\boldsymbol{w}}, \Sigma_{\boldsymbol{w}}\right) d\boldsymbol{w} = \mathcal{N}\left(y_t \mid \boldsymbol{\Psi}_t^T \mu_{\boldsymbol{w}}, \boldsymbol{\Psi}_t^T \Sigma_{\boldsymbol{w}} \boldsymbol{\Psi}_t + \Sigma_y\right) \tag{2.12}$$

Now, the mean and variance for any point t can easily be evaluated. As ProMP represents multiple ways to execute a specific movement, multiple demonstrations are needed to learn $p(w; \theta)$. The parameters $\theta\{\mu_w, \Sigma_w\}$ can be learned from these demonstrations by the maximum likelihood estimation for HBM with Gaussian distribution.

In fig. 2.4, a trajectory distribution of a single movement, generated using different approaches, is seen. Figure 2.4a shows the demonstrated trajectory distribution, which was generated by a stochastic optimal control algorithm for a via-point task. The variability in this distribution is due to the noise of the system. In constrast to the trajectory distribution generated using DMP, does the one generated using ProMP almost perfectly match the demonstrated distribution/

The example shown in fig. 2.4 presents a single movement distribution. In fig. 2.5, the implementation of ProMP for multiple goal positions is given. The left figure presents the distribution of fthe movements for five different target.

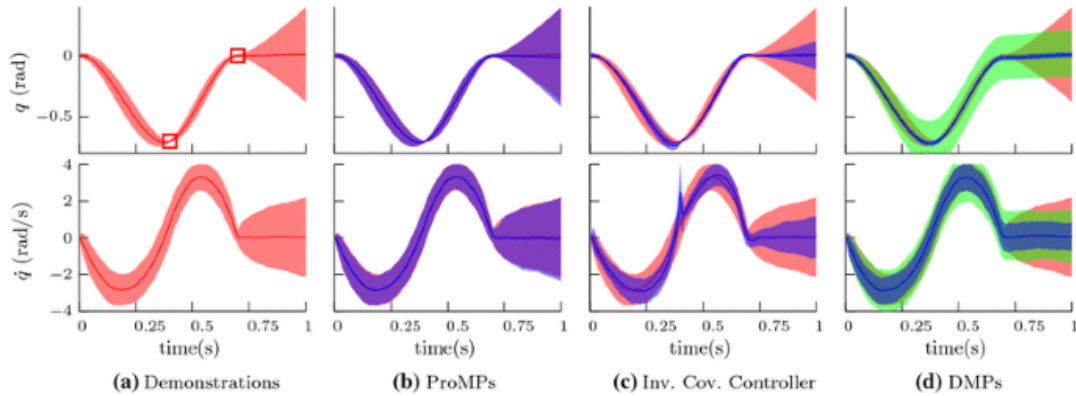**(a)** Demonstrations   **(b)** ProMPs   **(c)** Inv. Cov. Controller   **(d)** DMPs

Figure 2.4: The trajectory distribution, showing the joint position (upper plot) and velocity (lower plot). The shaded area is two times the standard variant. The red shaded area shows the demonstrated trajectory [30].
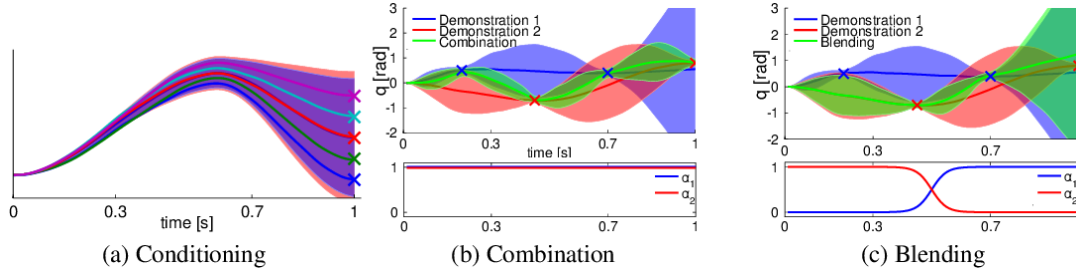
(a) Conditioning   (b) Combination   (c) Blending

Figure 2.5: The trajectory distribution encoded using ProMP for multiple goal positions. Here, two methods of combining the different demonstrations are indicated: combination and blending.

ProMPs are a relatively new topic of research in the literature. Therefore, not much research can be found implementing ProMP for online learning. However, one extension on ProMP seems to be promising: interaction ProMP. Interaction ProMP is a framework which can be used for collaboration with a human [33]. By taking inspiration from Interaction Primitives, ProMPs which realizes the primitives that capture the correlation between trajectories of multiple agents, is proposed. Interaction ProMPs uses the distribution over the trajectories of at least two interacting agents. Koert et al. use this principle in an online learning setting [34].

Gaussian Mixture Model and Gaussian Mixture Regression
Gaussian has received a lot of attention in literature. Therefore, it does not come as a surprise that this has also been frequently used for IL problems. Using just a Gaussian Regression method to solve IL problems would in most cases not provided enough complexity. Therefore, a combination of GMM with Gaussian Mixture Regression (GMR) is frequently used. This method is robust to noisy data, has a low computation cost, and is able to capture the correlation of human motion. Therefore, it is a powerful tool for robot analysis [35]. In this case, GMM is used to encode the behaviour of an expert and GMR to reproduce it.

Just as any other IL method, GMM requires a dataset $\mathcal{D}$, describing the motion of the demonstrations. Defining $\xi_t = (s_t, a_t)$, the joint distribution of s_{t+1} and ξ_t can be modded as a Gaussian distribution

$$p(s_{t+1}, \xi_t) = \sum_k p(k)\mathcal{N}(\mu_k, \Sigma_k) \tag{2.13}$$

where $p(k)$ is the prior. The kth Gaussian component can be described by

$$
\begin{aligned}
p\left(s_{t+1}, \xi_t \mid k\right) &= \mathcal{N}\left(\begin{bmatrix} \xi_t \\ x_{t+1} \end{bmatrix} \mid \begin{bmatrix} \mu_{\xi,k} \\ \mu_{x,k} \end{bmatrix}, \begin{bmatrix} \Sigma_{\xi,k} & \Sigma_{\xi x,k} \\ \Sigma_{x\xi,k} & \Sigma_{x,k} \end{bmatrix}\right) \\
&= \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}\left((\xi_t - \mu_k)^T \Sigma_k^{-1}(\xi_t - \mu_k)\right)},
\end{aligned}
\tag{2.14}
$$

where $\{\pi_k, \mu_k, \Sigma_k\}$ (the prior probability, mean vector, and covariance matrix) are the parameters of the Gaussian component k. Traditionally, such a model is iteratively trained using Expectation Maximization (EM). Other examples are spectral clustering, online learning or self-refinement. The learned model can then be used to reproduce movements [36]. An example of the encoding using GMM is visualized in fig. 2.6.

GMR can be used to reconstruct a general form for the signal [36]. GMR first models the joint probability density function of the data, and then derives the regression function from this model, which is different from other regression function which directly derive the regression function. In GMR, the estimation of the model parameters is done in the offline phase, making the regression independent of the number of data points. Therefore, the regression can be calculated very rapidly.
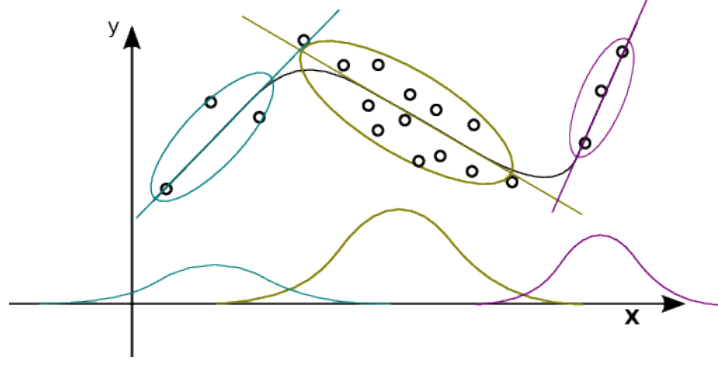
Figure 2.6: Example of trajectory encoding using GMM. The data points are indicated with a dot. The data points are clustered using three Gaussian components. Based on this, the Gaussian distributions are determined (bottom part of the plot), which results in a trajectory which is indicated with a grey, continuous line.

For each iteration step t, the data points can be composed into two parts, the temporal values, denoted with a t, and the spatial values, denoted with a s. This results in the following notation for the data points ξ, mean vectors μ, and covariance matrices Σ

$$\xi = \left[\begin{array}{c} \xi_t \\ \xi_s \end{array} \right], \quad \mu_k = \left[\begin{array}{c} \mu_{t,k} \\ \mu_{s,k} \end{array} \right], \quad \Sigma_k = \left[\begin{array}{cc} \Sigma_{t,k} & \Sigma_{ts,k} \\ \Sigma_{st,k} & \Sigma_{s,k} \end{array} \right] \tag{2.15}$$

For each component k, the expected distribution of $\xi_{s,k}$, given ξ_t, and the estimated covariance, can be determined:

$$\begin{aligned} \hat{\xi}_{s,k} &= \mu_{s,k} + \Sigma_{st,k} \left(\Sigma_{t,k} \right)^{-1} \left(\xi_t - \mu_{t,k} \right), \\ \hat{\Sigma}_{s,k} &= \Sigma_{s,k} - \Sigma_{st,k} \left(\Sigma_{t,k} \right)^{-1} \Sigma_{ts,k}, \end{aligned} \tag{2.16}$$

where $\xi_{s,k}$ and $\hat{\Sigma}_{s,k}$ are mixed according to the probability that the Gaussian component $k \in \{1, ..., K\}$, has been responsible for ξ_t. For a mixture of K components, the condition expectation of ξ_s, given ξ_t, and the conditional covariance of ξ_s, given ξ_t, can be described as

$$\hat{\xi}_s = \sum_{k=1}^{K} h_k \hat{\xi}_k, \quad \hat{\Sigma}_s = \sum_{k=1}^{K} h_k^2 \hat{\Sigma}_k, \tag{2.17}$$

where h_k is the probability of the component k to be responsible for t. An example of trajectory encoding using GMM and GMR is shown in fig. 2.7. This figure shows that at the beginning of the task execution, the constraints are loose (distribution has a high variance), whereas at the end it is quite strict.

In [37] two method for incrementally training the model were given. The first method, the direct update method, reformulates the problem for a generic observation of multiple data point. This is done using an adapted EM method, where the parts belonging to the already used data and those belonging to the newly available data are separated. The second method, the generative method, used EM performed on data generated by GMR.

Another paper [38] proposed three methods for incrementally updating a set of already existing trajectories, which were obtained using task-parametrized Gaussian Mixture Model (TP-GMM) [36]. TP-GMM in essence, models local (or relative) trajectories and corresponding local patterns, therefore endowing GMM with better extrapolation performance [39]. While these techniques do assume that previously trajectories were already obtained (either online or offline), this does show the ability to adapt in an online manner. The first technique estimates a new model by accumulating the new trajectory and a set of trajectories using the old model. The second techniques allow for adding parameters for the new trajectories, to the already existing parameters corresponding to the old trajectories. The last techniques update the model by usage of a modified EM algorithm, using the information of the new parameters.
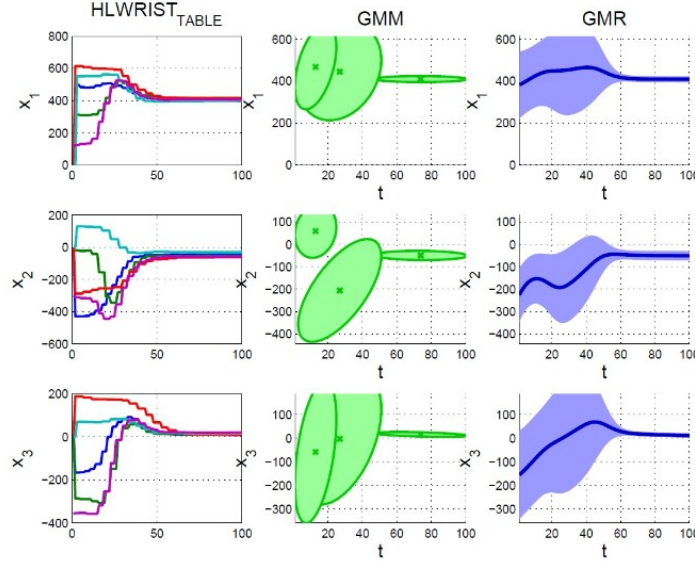
Figure 2.7: Trajectory encoding and generalization using GMM and GMR [35]. It is visible that at the start of execution the constraints are loose as the distribution has a high variance. In contrast, the final part shows that the constraints are quite strict as the variance of the distribution is low.

Cederborg et al. [40] proposed incremental local online GMR. This method allows for the robot to learn incrementally online a new motor task, by modelling them locally as dynamic systems. Sensorimotor context is used to cope with the absence of categorical information both during demonstrations and when a reproduction is asked of the system.

Gaussian Process
In the recent years, Gaussian Processes (GPs) as a solution for IL problems, has received more attention. GPs can be defined as a distribution over function, where inference directly takes place in the space of function [41]. GP aims to learn a deterministic input-output relationship, up to observation noise, based on a Gaussian prior over potential objective functions [42]. This is in contrast to GMR which provides a generative model. The difference between discriminative and generative models is graphically visualized in see fig. 2.8. While generative models, in most cases, require fewer data, their generalization performance is often poorer than that of a discriminative model.

GP models a mapping from the input $z = [x_t^T, u_t^T]$ to the output $x_{t+1} = f(z_t)$ as

$$f(z_t) \sim \mathcal{GP}(m(z_t), k(z_t, z_t')) \qquad (2.18)$$

where $k(z_t, z_t')$ is the covariance matrix [11]. Popular approaches for the covariance matrix are the squared exponential covariance function [42] and a Gaussian kernel [44]. The joint distribution of the given target value and the function value x_{t+1} at the test input z_t^* can be written as

Figure 2.8: Graphical representation of a generative and discriminative model [43].

$$\begin{bmatrix} x_{t+1} \\ x_{t+1}^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(Z,Z)+\sigma_n^2 I & K(Z,z_t^*) \\ K(z_t^*,Z) & K(z_t^*,z_t^*) \end{bmatrix}\right), \tag{2.19}$$

where Z is a matrix, where all input factor z_t of the training samples are combined. The mean μ and variance σ^2 are in this case dedicated by

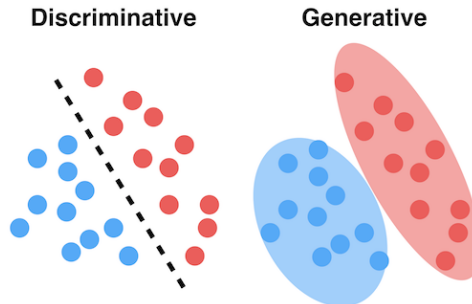$$\mu(z_t^*) = \boldsymbol{K}(z_t^*,Z)^\top (\boldsymbol{K}(Z,Z)+\sigma_n^2 I)^{-1} \boldsymbol{x}_{t+1} = \boldsymbol{K}(z_t^*,Z)^\top \alpha$$
$$\sigma^2(z_t^*) = k(z_t^*,z_t^*) - \boldsymbol{K}(z_t^*,Z)^\top (\boldsymbol{K}(Z,Z)+\sigma_n^2 I)^{-1} \boldsymbol{K}(z_t^*,Z), \tag{2.20}$$

where α denotes a prediction vector.

The explanation above presents the broad outline of GP. There have been multiple attempts to implement this approach for online learning. Nguyen-Tuong and Peters [44] proposed Local Gaussian Process (LGP), which combined the fast computation of local regression with a more accurate regression method. In [41] a GP-based learning approach was presented, which allows for generalization over multiple demonstrations, and encode variability along the different phases of the to be executed task. Jaquier et al. [42] proposed a GMR-based GP. This combination allows combining the advantages of GP to encode prior beliefs through the mean and kernel functions and of GMR to retrieve the variability information, to be encapsulated in the uncertainty estimated by the GP. Another advantage is that this approach has the properties of a generative model, which allows for new trajectories to be easily generated through sampling and conditioning. In [45] the Sparse Online Gaussian Processes (SOGP) is compared to Locally Weighted Projection Regression (LWPR). SOGP allows for data to be processed while it arrives. An advantage of this method, compared to the original Gaussian Process Regression (GPR) approach, is that it only a subset of the data and their associated kernel distances should be stored. Making the storage space much smaller than the one required for ~~GPR ($\mathcal{O}(N^2)$)~~.

Hidden Markov Model
The usage of Hidden Markov Model (HMM) for IL can be seen as a modified version of GMM, in which the choice of mixture component for each observation also depends on the choice of the component for the previous one. It allows for the modelling of a probabilistic transition between discrete states [11]. A reason why HMM is a useful representation of human skill, is because of its ability to discover the nature of the skill [46]. It does this by representing the training data in a statistic way, by its parameters, which allows for the retrieving of the skill model. This is of importance as it is expected that each demonstration, given by a human expert, will differ from the next.

HMM models are characterized by a Markov chain of sequence, consisting of (unobserved) hidden state variables s_t and a corresponding sequence of observation variable o_t [47]. A graphical representation of this is given in fig. 2.9. The different states are connected using a state transition matrix $A = \{a_i j\}$, where $a_{1,2}$ is the transition probability of state 1 to state 2. The output probability matrix is defined as $B = \{b_{ij}\}$, which denotes the probability of observing a certain output while being in a certain state. The initial state distribution is also defined and denoted by d. Now, the HMM can be denoted as a set of matrices $\lambda = \{A, B, d\}$ [46]. Given this set of matrices λ and an observation sequence $\boldsymbol{O} = \{o_1, \dots, o_T\}$, the likelihood of observing a given sequence $P(\boldsymbol{O} \mid \lambda)$, can be computed. The observed motion can then be found using

$$\lambda^* = arg \max_\lambda P(\boldsymbol{O} \mid \lambda). \tag{2.21}$$
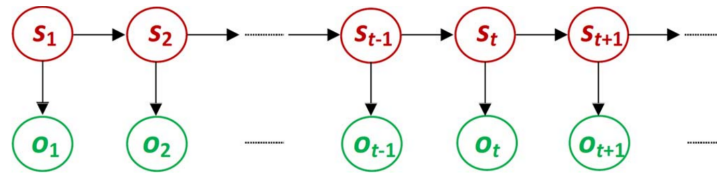

Figure 2.9: Graphical Representation of HMM [47]. With the s_n indicating the states and o_n the observations.

At the lowest level, each single MP is represented by a HMM. Given a set of MP $[\lambda_1, \ldots, \lambda_T]$ and an observation sequence $[O_1, \ldots, O_T]$, the state transition model and the probability distribution, can be learned, using the *Baum-Welch* algorithm [48].

To allow for incremental learning during observation, the system must be able to deal with new hidden states each time a new MP is learned. In addition, it should also have the ability to learn transition rules incrementally, from partial observations sequences. To do so, an incremental transition rule was proposed by Kulič et al. [48], which is applied each time a partial observation is available.

Using HMM allows for the encoding of the behaviour, demonstrated by an expert. There is, however, still need of a regression method which allows for the reproduction of this behaviour. One method for doings so is GMR [49–51]. An advantage of using this, in contrast to other regression methods such as LWR, LWPR or GPR, is that this does not model the regression function directly, but it models a joint probability density function of the data. After which it derives the regression function from the joint probability function. A drawback of the HMM representation is discreteness [11]. When the number of states is relatively low, HMM works well. Too few states can however result in the model not being able to reproduce a motion sequence. For robotic applications, this is a problem, as HMMs are often used to describe discrete, high-level states. To overcome this, other techniques combined with HMM can be used. An example of this is the state specific Gaussian models to represent continuous values [50].

In literature, some extensions of HMM have been found. Hidden Semi-Markov Model (HSMM) is defined as allowing the underlying process to be a semi-Markov chain [52]. A survey paper [52] identified multiple online implementations off HSMM, including an adaptive EM algorithm, an online algorithm base on recursive prediction error techniques, and a maximum likelihood estimation algorithm. Another extension is Hierarchy Hidden Markov Model (HHMM). In [53, 54], this framework is used to autonomously segments, cluster and learn sequencing of a full-body motion primitives from online observations of the full body human motion. This was expanded on in [48], where a higher abstraction HMM was used to learn higher level ordering between movement primitives. Lastly, Growing Hidden Markov Model (GHMM) [55] was proposed to learn motion patterns incrementally, and in parallel with prediction. While this method seemed to be a promising extension of the original HMM method, most research seemed to be limited to implementations for trajectory estimation for vehicles.

2.2.3. Inverse Reinforcement Learning Methods

Another form of IL is Inverse Reinforcement Learning (IRL). In IRL, the learner tries to recover a reward function of the environment, based on the expert's demonstration. Based on this reward function, it then tries to find the optimal policy using RL [7]. The used algorithm tries to find a reward which leads the learner to act similarly to the expert, while generalizing well to situations where the expert data is not available [6]. The assumption is made that the expert tries to maximize a certain reward function during execution of the demonstrations, making it the aim to estimate this function. Summarizing, IRL consist of two steps: 1) reward function estimation and 2) policy optimization based on this reward function. There are some variations to this general approach. For one, in some cases instead of finding a reward function, the goal is to find a cost function. This again makes the goal not to find a policy which minimizes this cost function. Another variation, found in literature [6], is to learn from failures instead of success. This approach is also referred to as Inverse Reinforcement Learning from Failure (IRLF).

The IRL problem was originally defined by Russel [56]. He described it as the following:

> **Given:**
> 1) measurements of an agent's behavior over time, in a variety of circumstances;
> 2) measurements of the sensory inputs tothat agent;
> 3) a model of the physical environment (including the agent's body).
> **Determine:** the reward function that the agent is optimizing.

Ng and Russel [7] defined three main types of IRL algorithms, which are still the most commonly used approaches [57]; 1) Finite-state MDP with known optimal policy; 2) Infinite-state MDP with known optimal policy; 3) Infinite-state MDP with unknown optimal policy, but demonstrations are given. The last

of these approaches is the closest to practical problems, as it seems more realistic that only the expert's demonstrations is available rather than the desired policy. Therefore, only this formulation will be considered. By obtaining data from an expert interacting with a MDP, this approach tries to find a reward function R^*, of which it was assumed that the expert was trying to maximize [6, 58]. This is the opposite off RL, which tries to find a policy that tries to maximize the expectation for a given reward function, by interacting with a MDP. Therefore, IRL is formalized as an *incomplete* MDP, also known as an MDP/R or in mathematically terms $\mathcal{M}/R$. This is a tuple of the form $\langle S, A, T, \gamma \rangle$ [58].

Most IRL approaches assume that there is a set of M features associated with every state, which fully determine the value of the reward function. As finding a general form solution for R can be challenging, most approach approximate it as a linear combination of the feature [7]. The reward function can then be defined as $R^* = w^{*T}\phi$. In literature, example have also been found where the reward function is not approximated using this linear approach, however, for now we will assume that it is. Assuming there is a feature mapping ϕ and the expert's feature expectation μ_E, the goal becomes finding a policy whose performance is close to that of the expert's. The policy to be found $\tilde{\pi}$ should be found, such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$. For such $\tilde{\pi}$, for any $w \in \mathbb{R}^k$ ($\|w\|_1 \leq 1$, the following should hold

$$
\begin{aligned}
\left| E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi_E \right] - E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \tilde{\pi} \right] \right| \\
= \left| w^T \mu(\tilde{\pi}) - w^T \mu_E \right| \\
\leq \|w\|_2 \|\mu(\tilde{\pi}) - \mu_E\|_2 \\
\leq 1 \cdot \epsilon = \epsilon.
\end{aligned}
\tag{2.22}
$$

Now the problem is reduced to finding a policy $\tilde{\pi}$, which induces feature expectation $\mu(\tilde{\pi})$ closest to μ_E. Here the policy can be found using policy iteration. A geometric description of policy iteration can be seen in fig. 2.10. The process of finding the policy $\tilde{\pi}$ can be described as:

1. Randomly pick some policy $\pi^{(0)}$ as initial policy, compute (or approximate via Monte Carlo (MC)) $\mu^{(0)} = \mu\left(\pi^{(0)}\right)$, and set $i = 1$.

2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T \left(\mu_E - \mu^{(j)}\right)$, and let $w^{(i)}$ be the value of w that attains this maximum.

3. If $t^{(i)} \leq \epsilon$, then terminate.

4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = \left(w^{(i)}\right)^T \phi$.

5. Compute (or estimate) $\mu^{(i)} = \mu\left(\pi^{(i)}\right)$.

6. Set $i = i + 1$, and go back to step 2.

A big problem in IRL is that multiple reward functions could explain the observations [59]. The reason for this is that the input is often a finite and small set of trajectories, resulting in many reward functions being able to realize the demonstrated data. Therefore, IRL suffers from an ambiguity solution. The literature often also refers to this problem as the result of IRL problems being "ill-posed" [7]. A method which tries to tackle this problem is maximum margin. It does this by converging on a solution which maximizes some margin. A downside of this method is that it introduces a bias into the learned reward function, thus resulting in exclusions of meaningful solutions [58] To avoid this, the maximum entropy
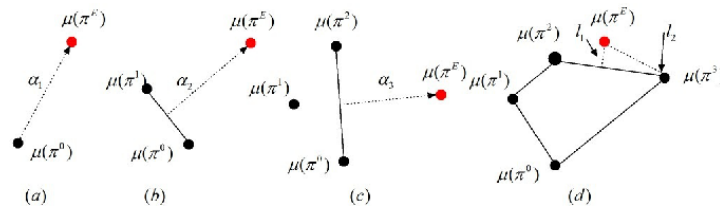

Figure 2.10: Geometric description of policy iteration. The goal is to find a feature expectation $\mu(\tilde{\pi})$, with a different smaller than ϵ compared to the expert's feature expectation μ_E. This difference is indicated as l_1.

principle could be used. This approach is attractive as it is probabilistic and thus robust to noise and randomness in the demonstrations of the expert [6]. In the remaining of this section, both methods will be discussed.

Max-Margin Inverse Reinforcement Learning
In maximum margin IRL [58], the reward function is identified by maximizing the difference between the best policy and all other policies [9]. The maximization step could be written as the following [58]

$$\min_{t,w} \quad t \tag{2.23a}$$

$$\text{subject to} \quad w^T \mu_E \geq w^T \mu^{(j)} + t, j = 0, \dots, i-1 \tag{2.23b}$$

$$\|w\|_2 \leq 1. \tag{2.23c}$$

This shows that the goal is to find a reward function $R = w^{(i)}\phi$, where the expert does better, by a margin of t, than any of the policies previously found. It assumes that there is already an initial estimate of the reward function $j = 0$. The constraint shown in eq. (2.23c) shows a 2-norm constraint, which means that the problem cannot be seen as a linear program, but only as a quadratic program [58].

The apprenticeship learning problem as presented in [58] had as downside that there was no mechanism for explicitly matching to the experts' behaviour. In addition, the solution was a stochastic mixture of multiple policies [60]. Ratliff, Bagnell, and Zinkevich therefore proposed maximum margin planning (MMP) [61], to address these problems. This frequently used approach, produces a single deterministic solution while also ensuring an upper bound on the mismatch between the demonstrated and planned behaviour. The research of Ratliff, Bagnell, and Zinkevich also provides an extension of MMP for online learning, where they also prove that the algorithm holds to a sublinear regret bound.

While the first research on maximum margin IRL does provide a framework for online IRL, not much research has been found where this method has been implemented for online learning. The main reason for this is probably the existence of another IRL approach, namely maximum entropy IRL [62]. The maximum margin approach [58] has a major downside that the biases, which help to search in the ill-posed problem, can also rule out some other meaningful solution. As the maximum entropy approach makes fewer assumptions, it is often desired over maximum margin.

Maximum Entropy Inverse Reinforcement Learning
As the name states, maximum entropy uses the maximum entropy principle to compute a reward function which is best represents the demonstrated behaviour [62]. This allows for obtaining a distribution over behaviour, which are parameterized by the reward function [11]. By using a probabilistic approach, it tries to solve the IRL problem [57]. The probability of an observed experts' trajectory τ is weighted by the estimated reward. This results in policies with a higher reward, to be exponentially more preferred:

$$p\left(\tau \mid w\right) = \frac{1}{Z(w)} e^{w^T \phi(\tau)} \tag{2.24}$$

where $Z(w) = \sum_\tau \exp(w^T \phi(\tau))$ is the partition function. This expression of the probability of the trajectory, however, only holds for deterministic environments. In the case of stochastic environments, this probability is also affected by the transition probability:

$$p(\tau \mid w) = \frac{1}{Z(w)} \exp\left(w^\top \phi(\tau)\right) \prod_{x_{t+1}, u_t, x_t \in \tau} p\left(x_{t+1} \mid u_t, x_t\right). \tag{2.25}$$

The following function is now tried to be optimized:

$$\tilde{R}(\tau) = \boldsymbol{w}^\top \phi(\boldsymbol{\tau}) + \sum \log p\left(\boldsymbol{x}_{t+1} \mid \boldsymbol{u}_t, \boldsymbol{x}_t\right) \tag{2.26}$$

Visible is that there is now a bias term due to the stochasticity of the environment, which is the main (theoreticial) drawback of this approach. This has, however, been adressed by the maximum causal entropy IRL.

The optimal value of the parameter vector w of the reward vector R, is given by maximizing the likelihood of the observed trajectory through maximum entropy:

$$w^* = \underset{w}{\text{argmax}} \; \mathcal{L}_{ME}(w) = \underset{w}{\text{argmax}} \sum_{\tau^{\text{demo}}} \ln p\left(\tau^{\text{demo}} \mid w\right) \tag{2.27}$$

where $\mathcal{L}_{ME}(w)$ describes a convex, objective function. Due to its convexity, this allows for solving the problem using a gradient-based method [11]. The gradient is then given as the empirical feature counts from demonstration and the expected feature counts from the learner's policy:

$$\nabla \mathcal{L}_{\text{ME}}(\boldsymbol{w}) = \mathbb{E}_{\pi^{\text{E}}}[\phi(\boldsymbol{\tau})] - \sum_{\tau} p(\boldsymbol{\tau} \mid \boldsymbol{w})\phi(\boldsymbol{\tau}) = \mathbb{E}_{\pi^{\text{E}}}[\phi(\boldsymbol{\tau})] - \sum_{x_i} D_{x_i}\phi\left(\boldsymbol{x}_i\right) \tag{2.28}$$

Maximum entropy IRL works well for MDP problems, as it assumes that the state transition distribution is known [11]. However, this is often not the case in many robotic applications. Therefore, sampling-based or model learning extensions must be applied for problems where the model is unknown. In [63] such an approach was used for autonomous driving. The algorithm allowed for directly learning the reward function in the continuous domain, while also considering the uncertainties in the expert's demonstrations.

Arora, Doshi, and Banerjee [64] proposed a framework online IRL called incremental Inverse Reinforcement Learning (I2RL). Based on this, max-entropy [62] was adapted such that it was generalized under occlusion. The results showed similar performance to the state-of-the-art applications of maximum-entropy IRL, while significantly reducing the computational time.

2.3. Reinforcement Learning

Reinforcement Learning (RL) enables learning through interaction. The main principle of RL has been represented in a flow diagram fig. 2.11. By interacting with the environment and observing the consequence of its action, and agent can learn to change it behaviour based on the rewards it has received [65]. Based on this trial-and-error approach, the agent is able to autonomously discover an optimal behaviour for a certain task [66], also called the policy. As the agent will learn of its environment, there is also no need of an external teacher. The need for the agent to interact with its environment in order to learn, is the main difference between RL and other types of Machine Learning [67]. This makes RL methods able to deal with changing environments. ~~For example, in Supervised Learning, the agent is presented with a sequence of (independent) examples of correct predictions, and in imitation learning the agent is provided with a demonstration of actions for a certain task [66].~~

While the principle of RL used for robotic applications is the same as the general approach, applying it comes with some challenges [66]. Firstly, learning takes place in a high-dimensional space (10 to 30 dimensional space). In addition, the states and actions are also continuous. This results in the learning
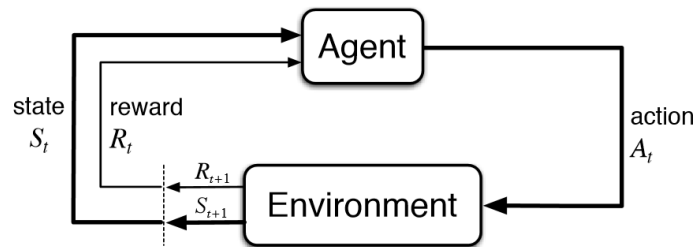

Figure 2.11: The agent-environment interaction in reinforcement learning as defined by Sutton & Barton [68]

being very costly. Secondly, it is unreasonable to assume that the states are completely observable and noise free. Therefore, the problem is often seems as partially observable. Thirdly, the algorithm needs to be robust in respect to models that do not capture all the details of the real system, also referred to as under-modelling, and to model uncertainty. And lastly, RL algorithms require a well-defined reward system. Therefore, it requires a fair amount of domain knowledge and may often be hard in practice. A solution for this problem is to use IRL to learn the reward function and RL to maintain the desired trajectory or to perform the desired task, once the learning has been completed. A description of IRL was already given in section 2.2.3.

There are two widely used approaches which can be used to obtain the solution of a RL problem: the *value function* approach and the *policy search* approach [67]. In addition, a new approach has been developed called actor-critic. This approach combines the value function and policy search method. The remaining of this section will first consist of explaining the general problem definition in section 2.3.1. Next, the three categories of RL methods will be discussed in sections 2.3.2 to 2.3.4.

2.3.1. Problem Definition Reinforcement Learning

In RL, the agent tries to maximize the accumulated reward (or minimize the cost) over a specific time. It does this by using a trial-and-error approach. This allows for the agent to learn tasks which could not be demonstrated or programmed by a human expert. The goal of RL is to find the π^* which maps the states (or observations) to actions, such that expected return J (defined by the reward function R) is maximized. Different expressions for the expected return have been defined. The main types of function indicate that the agent's return can be either discounted or averaged, and are calculated on a finite or infinite horizon [67]. In discounted, RL the choosing of the discount factor is critical. Choosing this value too low can result in an unstable behaviour, making it often not suitable for robotic application. Average rewards have the problem that it cannot distinguish between policies that initially gain a transient of larger rewards, and those that do not. However, this disadvantage is far less critical than the one of the discount factor, thus making the average reward setting often more desirable for robotic applications. The average reward return is then defined as:

$$J(\pi) = \sum_{s,a} \mu^\pi \pi(s,a) R(s,a), \tag{2.29}$$

where μ^π is the stationary state distribution generated by policy π acting on the environment, i.e. the MDP. As classic RL approaches, are based on the MDP, in theory they should only work when it adheres to the Markov structure. However, it has also been discovered that in reality the approaches often still work when this is not the case [66]. The finding of a policy π can now be described as an optimization problem for $J(\pi)$. This problem has been formulated as an optimization problem

$$\max_{\pi} J(\pi) = \sum_{s,a} \mu^\pi(s)\pi(s,a)R(s,a), \tag{2.30a}$$

$$\text{s.t. } \mu^\pi(s') = \sum_{s,a} \mu^\pi(s)\pi(s,a)T(s,a,s'), \forall s' \in S, \tag{2.30b}$$

$$1 = \sum_{s,a} \mu^\pi(s)\pi(s,a), \ \pi(s,a) \geq 0, \forall s \in S, a \in A. \tag{2.30c}$$

Here, eq. (2.30b) defines the stationarity of the state distribution π, which ensures that it is well-defined, and eq. (2.30c) ensured the proper state-action probability distribution. To search for a solution originally two approaches were used. Firstly, the optimal solution can be searched directly in its original, primal problem, which is known as policy search. Secondly, it can be optimized using the Lagrange dual formulation, which is known as a value-function based approach. In addition, in the recent years a third approach has been developed: the actor-critic approach.

Besides the types of algorithm, there are two more criteria on how to categorize RL problems: model-based or model-free, and on-policy or off-policy [5, 69]. An overview of the taxonomy of RL methods is given in fig. 2.12.
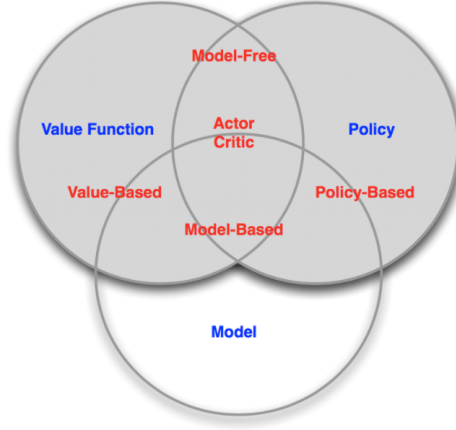
Figure 2.12: Taxonomy of RL methods [70].

Just like in most types of learning methods, both model-based and model-free methods exist for RL. Here the model-based methods rely on the model of the environment which is either known or can be explicitly learned to use the algorithm. In contrast, do the model-free methods not depend on such a model [66]. The advantage and disadvantage were described by Polydros et al. [67] and are shown in table 2.4.

The difference between *on-policy* and *off-policy* method is that the former tries to evaluate or improve the policy to make decisions, whereas the latter does it to generate the data [66]. On-policy require the agent to interact with the environment, meaning the policy that interacts with the environment should be the same as the one which has to be improved. In the off-policy, this is not necessary the case. For example, the experience of other agents can also be used to improve the policy.

In the remaining of this section the different approaches (value-function, policy search (PS) and actor-critic) will be discussed. Here both a general notation and some classes of methods will be described.

2.3.2. Value-function approaches

This approach tries to find the optimal policy, by iteratively optimizing the value function. It assumes that each state has an associated value dependent on the reward achieved in that state. Each state also has a potential, which depends on the future reward achieved using the agent's current policy [71]. As it is often the case that a reward is only received in a specific goal state, the positions closest to this state have the highest value. By breaking down the problem into sub-problem, following the sequence of states, the value function can be found.

Value-function methods try to find a solution to the condition for optimality. This depends on the Lagrange multipliers $\bar{R}$ and $V^\pi(s')$, and is given by

$$V^*(s) = \max_{a^*} \left[(R(s, a^*) - \bar{R} + \sum_{s'} V^*(s') T(s, a^*, s') \right],$$ 					(2.31)

Table 2.4: Advantages and disadvantage of model-free and model-based RL methods [67]

|	Advantages	Disadvantages
Model-based	• Small number of interactions between robot & environment • Faster convergence to optimal solution	• Depend on transition models • Model accuracy has a big impact on learning task
Model-free	• No need for prior knowledge of transitions • Easily implementable	• Slow learning convergence • High wear & tear of the robot • High risk of damage
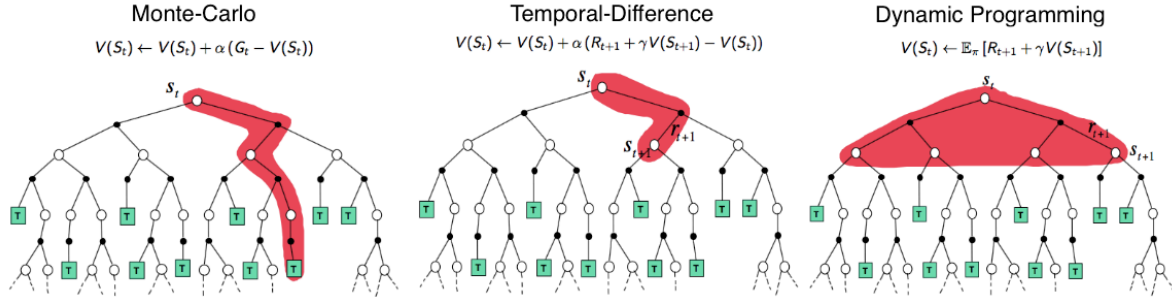
Figure 2.13: Comparison of the backup diagrams of MC, TD learning, and Dynamic Programming for state value functions.

where $V^*(s)$ is the shorthand notation of $V^{\pi^*}(s)$. This formulation is equivalent to the *Bellman principle of optimality*. This approach owes its name to the value function $V^\pi(s)$. Instead of using this function, another option is to use the state-action value $Q^\pi(s, a)$. As its name suggest, in addition to the state (where the value function depends upon), does this value also depend on the action. The different methods which have attempted to estimate $V^*(s)$ or $Q^*(s, a)$, can be divided into three classes: Dynamic Programming (DP) based optimal control approaches, rollout-based MC methods and Temporal-Difference (TD) methods. A graphical representation of these methods is given in fig. 2.13. These classes will next be discussed.

Dynamic Programming based methods
DP based methods, use a model of the transition probabilities $T(s', a, s)$ and the reward function $R(s, a)$ to calculate the value function, making it a model-based approach. Two well known approaches are policy iteration and value iteration [66]. A graphical representation of both method is shown in fig. 2.14.

Policy iteration starts with initializing an arbitrary policy [66]. After this, it alternates between two phases: policy evaluation (state or action function is calculated for a given policy) and policy improvement (the best action for each state is derived) [67]. In the evaluation phases, the value function for the current policy is evaluated. Here, each state is visited, and its value is updated based on the current value estimates of its successor states, the associated transition probabilities, as well as the policy. This procedure is repeated until the value function converges to a fixed point, which corresponds to the true value function. Next, in the improvement phases, an action is greedily selected in every state according to the value function. These two steps are repeated until the policy does not change any longer [66]. A disadvantage of policy iteration is that the iteration can be a time-consuming task [67].

In contrary to the policy iteration method, where the policy is only updated after the evaluation step has converged, the value iteration method does not wait until the convergence of the evaluation procedure for updating the policy. The value function is updated this based on eq. (2.31) every time a state is updated [66, 67].
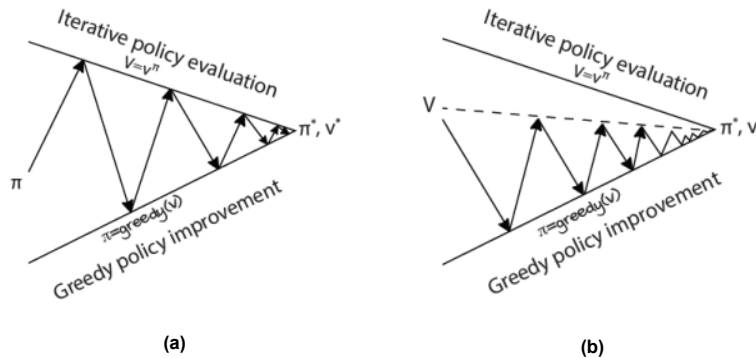

Figure 2.14: Graphical representation of the DP methods. With *(a)* the representation for policy iteration and *(b)* the representation for value iteration.

Monte Carlo (MC)

In contrast to the DP approach, is this method model-free [72]. It can be used to replace the policy evaluation step of the value or policy iteration method [66]. The value function V^π can be estimated by randomly sample many trajectories starting from state s, according to the given state transition matrix P [72]. This is done by keeping track and using the frequencies of the transitions and rewards. In the paper of Ding et al. [72] an example of this method is given. MC perform roll-outs by executing transition function, in other world operating on-policy [66].

Temporal-Difference (TD)

This method combines the ideas from DP and MC to estimate the value function [72]. Like DP, TD uses bootstrapping in the estimation. This is used to form a target from the observed return and an estimated state value for the next state [72]. On the other hand, just like MC, a sample approach is used to estimate the value function. To be precise, sample transition in the MDP are used [66]. To learn, TD uses the error (difference between target value and estimated value) at different time steps [72]. A basic TD method for the update is

$$V' = V(s) + \alpha \left(R(s, a) - \bar{R} + V(s') - V(s) \right), \tag{2.32}$$

with $V(s)$ the old estimate of the value function, $V'(s)$ the new one, and α the learning rate. This method is also referred to as the TD(0) or one-step TD method [66, 72], as it looks only one step ahead. N-step TD can also be developed by extending the target value with discounted rewards in the N-step future and estimate state value at the N-th step. Famous TD methods are `SARSA`, which is on-policy, and `Q-learning`, which is off-policy.

~~**Comparison**~~

~~Looking at the different methods using the value function approach, TD can be seen as method in between DP and MC method. TD and DP have as similarity that they both use bootstrapping estimation, which means that estimating the value of the current step is done using an estimation from other steps, e.g. from the next. MC can instead estimate different states independently of each other, which result in less bias, but higher variance [72]. This result is also enabled due to the fact that MC has to wait unital an episode is finished to update, whereas DP can update at each time step [69]. A similarity between TD and MC is that they do not require full knowledge of the environment.~~

2.3.3. Policy Search

A problem with value function approximation is that it result in a difficult problem in high-dimensional state and action spaces, which can be problematic for robotic applications. Another problem is that value functions are often discontinuous, which result in the propagation of error in the value function through to the policy [73]. Therefore, PS methods could provide a good alternative. In these methods, the optimal policy is learned directly [67]. Generally, they exist of a set of parameters, which has to be optimized such that the cumulative reward is optimized. Where value function search in the state-action space for the best policy, does PS directly search in the parameter space. This allows for scaling RL into high-dimensional continuous action space, by reducing the search space of possible polices. Which, in other words, means that it is less occupational expensive. PS has been seen as a more difficult approach, is comparison to value function, as an optimal solution cannot be directly obtained using eq. (2.30). However, PS has retrieved popularity in the field of robotics, due to better scalability and the convergence problems of the approximate value-function method [66].

PS, like all RL methods, can be categorized into model-free and model-based methods. Here, the model-free methods use the real robot interactions to create sample trajectories. Whereas, the model-based policy method attempts to address the problem of sample inefficiency by using the observed trajectory to learn a forward model of the robot's dynamics and its environment. Model-based method can again be divided in stochastic trajectory generation, which (just as model-free) uses sampled trajectories, and deterministic trajectory generation, which analytically predicts the sample distribution. The latter is computationally more expensive, however it analytically computes gradients, whereas the stochastic approach can only estimate this.

The policy update strategies for both model-free and model-based methods are based on either policy

gradients (PGs), EM, or information-theoretic insights (Inf.Th.) [66, 67, 73]. These will further be explained in this section.

Most policy-search method optimize locally around existing policies π, parameterized by a set of policy parameters θ_i. This is done by computing changes in the policy parameters $\Delta\theta_i$ which will increase the expected return and results in iterative updates of the form

$$\theta_{i+1} = \theta_i + \Delta\theta_i. \tag{2.33}$$

The key step is the computation of the policy update. ~~There is a variety of updates proposed which range from pairwise comparisons over gradient estimation using finite policy difference and general stochastic optimization methods, cross entropy and population-based methods, to approaches coming from optimal control such as DDP and multiple shooting approaches.~~

Looking at the different policy update strategies, a distinction between the two evaluation strategies, the step-based and episode-based evaluation strategy, should be made.

Policy gradient ~~(PG)~~
The gradient-based approaches, use a gradient ascent for maximizing the expected return J_θ [66, 73]. Here, the policy is updated following the gradient of the expected return $\nabla_\theta J_\theta$ for a defined step size α. For iteration $i+1$, the policy is given by

$$\theta_{i+1} = \theta_i + \alpha\nabla_\theta J_\theta. \tag{2.34}$$

The policy gradient can then be computed using

$$\nabla_\theta J_\theta = \int_\tau \nabla_\theta p_\theta(\tau)R(\tau)d\tau. \tag{2.35}$$

Different approaches exist to compute the gradient, and many algorithms also require tuning the step-size α. Three approaches to compute the gradient are: finite difference gradient, the likelihood ratio method, and natural gradient.

The finite difference gradient is the simplest approach and is typically used in an episode-based setting. This method computes the gradient by applying a small perturbation $\delta\theta_p$, to the parameter vector θ. By using linear regression, the gradient can now be estimated. While this is a straightforward approach and even applicable for non-differentiable policies, it often considered to be noisy and inefficient.

The second approach uses the likelihood ration trick, to compute the gradient of the episode distribution. By inserting the gradient of the episodic distribution $\nabla_\theta p_\theta(\tau) = p_\theta(\tau) \log p_\theta(\tau)$, in eq. (2.35), the policy gradient can be computed. Here $p_\theta(\tau)$ is approximated using the sum of the sampled trajectories τ. Algorithms using this approach is `REINFORCE` and `G(PO)MDP` [73].

As convergence of the finite difference gradient and likelihood ratio method seemed to be kind of slow, the natural gradient policy approaches, which allow for fast convergence, might be advantageous for robotic application.

Expectation Maximization ~~(EM)~~
PG method often require the setting of a learning rate, which can be problematic and result in unstable learning or slow convergence. This problem can be avoided by formulating the policy search as an inference problem with laten variables and using the EM algorithm to derive the new policy. Some of the approaches which have been proven successful in robotics are: reward-weighted regression, policy learning by weighted exploration with returns, MC EM, and cost-regularized kernel regression.

Information-theoretic insights (Inf.Th.)
This last approach tries to update the parameters by "staying close" to the provided data. This means that the trajectory distribution after the policy update should not jump away from the trajectory distribution before the policy update. Therefore, Inf.Th. approaches bound themselves by the old trajectory distribution $q(\tau)$ and the newly estimated trajectory distribution $p(\tau)$, at each update step. This limits the information loss of the updates, and thus avoids that the new distribution convergence to a local optima. Some method which have applied this idea, are the natural policy gradient algorithm and the Relative Entropy Policy Search,

2.3.4. Actor-critic method

The actor-critic (AC) approach form a special class of PG methods, as they aim to combine the advantage of both the value function and PS approaches. The idea behind this method is to separate the policy (actor) and value function (critic) as visible in fig. 2.15. Both of these entities are represented as a parametrized function of the state x, with the parameters ψ and ϑ, respectively. At time step t, the actor calculates an action u_t based on the current state x_t and applies it to the system. This results into the transition of to a new state x_{t+1} and a reward r_{t+1}. Next, the evaluation of policy can be done using any policy evaluation method. The method explained in [74] use TD for this. The TD-error δ can be calculated based on the new state and reward. This error is used to estimate the actual value function and criticize the actor, by updating the corresponding parameters. If $\delta > 0$, the action is favourable which means that the actor should go toward the direction. In contrast, if $\delta < 0$, the actor should avoid taking that action in the next step.
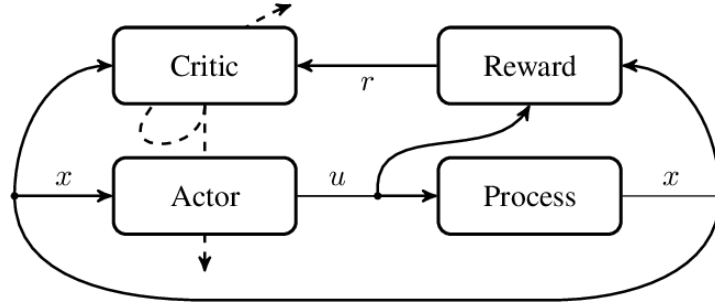

Figure 2.15: Schematic overview of an actor-critic algorithm [74].

Algorithm 2 shows the algorithm of the actor-critic method. While most points can be connected to the image in fig. 2.15, some additional explanation is required to understand this algorithm. First some coefficients are initialized, which represent are the trace decay rate $\lambda \in [0, 1)$, discount factor γ, the learning rate for the actor α_a and the learning rate of the critic α_c. The next line initializes the actor and critic parameters. To increase the probability of visiting all the system states multiple times, an exploration signal Δu_t is added to the output of the actor. A standard way to update the critic is to use the TD-error δ. If the value function is approximated to be linear $V_\theta(x) = \theta^T \phi(x, u)$, then the TD-error can be written as $\delta_{t+1} = \delta_t + \alpha_{c,t}\delta_t\phi(x_t)$. This is also referred to as $TD(0)$, as no eligibility traces are used. In algorithm 2, however, the extension is shown where an eligibility trace ζ_t is used (line 14). This causes the critic to be updated based on a series of visited states instead of just the prior one, which should have the advantage that the learning is speed up. Finally, to update the parameters of the critic and actor, the TD-error is used.

Grondman et al. [74] defined two criteria for the taxonomy of AC method. Firstly, just as in other types of RL methods, a distinction between discounted return and average return can be made. In addition, two types of gradients have been identified: standard and natural.

2.4. Combining Imitation Learning and Reinforcement Learning

IL and RL both provide features which makes them desirable for robotic teaching. As stated before, IL can fast learn a certain skill, however it is limited by the performance by the skill of an expert. This problem does not occur for RL, however, its learning time is often much slower. The combination of these two methods would therefore be preferable. In this case, both best features could be combined:

Algorithm 2 Actor-critic Algorithm [75]

1: Initialize λ, γ, α_a, α_c
2: Initialize ϑ_0, ψ_0
3: **for** each trial **do**
4:      Initialize x_0
5:      Generate a random initial action u_0
6:      Initialize eligibility trace $\zeta_0 = 0$
7:      $t \leftarrow 0$
8:      **repeat**
9:          generate exploration Δu_t
10:          calculate current action $u_t = \hat{\pi}\left(x_t, \psi_t\right) + \Delta u_t$
11:          apply u_t, measure x_{t+1}
12:          receive reward $r_{t+1} = \rho\left(x_t, u_t\right)$
13:          $\delta_t = r_{t+1} + \gamma \hat{V}\left(x_{t+1}, \vartheta_t\right) - \hat{V}\left(x_t, \vartheta_t\right)$ $\triangleright$ calculated TD-error
14:          $\zeta_{t+1} = \lambda\gamma\zeta_t + \left.\frac{\partial \hat{V}(x,\vartheta)}{\partial \vartheta}\right|_{x=x_t, \vartheta=\vartheta_t}$ $\triangleright$ eligibility trace
15:          $\vartheta_{t+1} = \vartheta_t + \alpha_c\delta_t\zeta_{t+1}$ $\triangleright$ update critic parameter
16:          $\psi_{t+1} = \psi_t + \alpha_a\delta_t\Delta u_t \left.\frac{\partial \hat{\pi}(x,\psi)}{\partial \psi}\right|_{x=x_t, \psi=\psi_t}$ $\triangleright$ update actor parameter
17:          $t \leftarrow t + 1$
18:      **until** t = maximum number of samples T_s
19: **end for**

---

use the expert to quickly learn a reasonable policy by imitation and exploring how to improve the skill upon the expert by using RL.

Chang et al. tried to implement the Locally Optimal Learning to Search (LOLS) algorithm, which combines IL and RL by stochastically interleaving incremental RL and IL updates [76]. For doing so, the IRL framework learning to search (L2S) was used in combination with a policy-iteration based RL method. As a result, the learned policy would either perform as well as the expert policy (due to IL method) or reach a local optima (due to RL method). While it is possible that the local optima result in a better performance than the experts' policy, it is difficult to quantify this.

In [77], the idea of combining IL and RL was implemented via the idea of reward shaping. For the IL part, access is assumed to a cost-to-go oracle, which provides an estimate of the expert's cost-to-go during training. The principle of cost-to-go (or reward-to-go) is that a cost is only received after a certain action has been taken. After shaping the cost, the planning horizon of the new MDP would be truncated, followed by a search for a policy that optimizes over the truncated planning horizon.

While this idea of combining the two learning method seems to be promising, the literature seems to be limited. The only robotic implementation found, was exterted by Sun, Bagnell, and Boots, who implemented their approach in an online robotic simulation of OpenAI Gym.

2.5. Comparison

In this section, a comparison of the different learning approaches, described earlier in this chapter, will be made. This comparison is shown in table 2.5. Here, the different robot teaching methods are compared based on multiple criteria. This comparison is a summary of information found in literature and made assumptions based on this. The literature used for this, will be described in the remaining of this section. In order to have a more reliable comparison, further research should be conducted.

The *computational efficiency* describes the amount of data and computational time, required to converge to a solution [9]. Especially for real-time application, this is of the highest importance as a high computational cost would result in a slower system, thus making this system less reactive. Slow learning might not necessarily be a problem when learning in a "save" environment, it could be a problem when adaptation to the environment is required. IL method often have a higher computational efficiency than RL. This is due to the fact that fewer iterations are needed to obtain the policy. RL on the other hand, use a trial-and-error approach, resulting in a much more iteration. Looking at the different IL

Table 2.5: Comparison of robot teaching methods. It should be noted that this comparison is based on an interpretation of literature. In order to have a more conclusive analysis, research is required. The methods are compared based on some criteria, here + indicates a high score relating to the criteria, +/- an average score, and - a bad score. These score were given with respect to another. From left to right the different criteria are; *Computational efficiency* describing the amount of necessary to learn. *Stability* describes the ability to deal with perturbation. Smoothness refers to the smoothness of the trajectory. Generalizability describes the ability of the system to deal with situation not occurred during the learning process (e.g. new starting point or goal). The difficulty of implementing a certain system is indicated by *implementation*. *High-dimensional input* states whether a system can deal with large amount of data. Lastly, *online* describes whether the system were originally implemented for online usage.

	Method	Comp. efficiency	Stability	Smoothness	Generalizability	Implementation	High-dim. input	Online
IL	BC	+	+/-	+	+	+	+/-	-
	IRL	+/-	+/-	+	+	+/-	+	+
RL	Value function	+/-	- [1]	+/-	-	-	-	+
	Policy Search	-	-	+/-	-	-	+	+
	Actor-Critic	-	+/-	+/-	+/-	-	+	+

[1]*No information about the stability of value function method could be found, so this is an assumption based on the information of the other RL methods.*

method, BC often only have to do one iteration per demonstration. There is, however, a difference in computational time between the different BC method. ~~So, does~~ DMP just have to update one parameter. Whereas, GP has to compute a new covariance matrix (which can be large for high-dimensional inputs) and has to invert this. As IRL typically have to update their cost function multiple times, their sample efficiency is slightly worse than those of the BC. Looking at the RL methods, value function are known to have a high sample efficiency in comparison to the PS method [5]. This could be translated in the former having a higher computational efficiency than the latter. As actor-critic methods uses the bootstrapped value-function to reduce the variance of the gradient estimate, they gain the sample efficiency of the value function.

Stability refers to the ability of the system to deal with perturbations. In cases where perturbation would result in big difference in the behaviour, the system said to be unstable. Instead, it is desired that the behaviour only changes a small fraction or maybe not even at all. This is desired to ensure a safe design, which is of importance to avoid a robot damaging itself, and something or someone else. The stability of the BC methods is not always ensured. ~~So, is~~ DMP statically asymptotically stable [78], meaning it has a stable attraction to a target position. Whereas ProMP [11], GMR and GPR [79] do not provide this guarantee. As RL methods generally search a large part of the solution space to find a policy, it is assumed that they generally have a good stability. However, this is not necessary the case as most method depend on multiple conditions. ~~So does~~ PG require the setting of a learning rate, which, if chosen incorrectly, results in unstable behaviour [73]. This ~~instability~~ problem was, however, tackled by formulating PS as an inference problem with latent variables and, using the EM algorithm, to determine a new policy. The Inf.Th. principle states that in order to provide a stable behaviour, the distance between the old and new trajectory distribution, should be bounded. A method which has been identified to increase stability is the usage of replay [71]. During replay, the robot is initiated in states which are not stored, in order to improve exploration efficiency. This also has the effect of becoming more stable. As the actor-critic algorithm `A3C` allows for multiple agents to asynchronously explore different policies in parallel, the usage of replay was not require to enesure stability.

With *smoothness*, the smoothness of the encoded behaviour (often trajectory) is meant. To minimize the risk of damage to the robot, it is desired that the behaviour is smooth, without sharp changes [9]. This criterion is hard to quantify as it has to be visually examined. Therefore, results become quite subjective. For the IL approaches, the different smoothness was based on literature. For each approach, multiple papers were used (for BC [31, 37, 41, 50, 80, 81] and for IRL [61, 82]) ~~to determine~~

the smoothness of the method. As IL methods try to imitate the expert, which is assumed to have a smooth behaviour, the resulting behaviour of the robot is in most cases also smooth. The same could not be said for RL method. Here, the smoothness of the system highly depends on the chosen reward function. Just as for the stability, the usage of a bad trajectory would result in a rough behaviour.

Generalizability is the ability of a system to perform well in similar environments as used for the training of the system [59]. Examples of this are states and action unobserved in the demonstration, starting the task at a different initial state, and reusing skills in different problem settings [5]. BC method often generalize quite well. However, in some cases some extensions are required [11]. An example is ProMP which learn the distribution of demonstrated trajectories in parameter space. To generalize to a new start and goal position, the learned distribution has to be conditioned. An example of this was given in [81]. Research has also proven IRL method to generalize well to un-encountered settings [62]. Generally speaking, RL often has a lack of generalizability [71]. A solution could be to increase the sample dataset and train on multiple (random) environments [83]. This, however, comes at the cost of computational efficiency. A difference can also be seen between model-based and model-free approaches. Where the former often generalizes better compared to the latter [71]. There have been PS algorithm which generalize relatively well. Most of these are episodic-based policy evaluation strategies [73]. It should be noted that research on generalizability for RL, still seems to be limited.

It is desired that a system is easy to implement. Difficult *implementation* can mean that a lot of time needs to be spent to implement it, and in addition, that more can go wrong implementing it. For example, as more assumptions have to be made. Overall, BC methods are quite easy to implement. The reason for this is that most methods are model-free [11]. In contrast, IRL methods are often model-based, meaning that if no model of the system is available, implementation can become a lot more difficult. The implementation of RL the method is often more difficult than implementing IL. Besides the difficulty of determining a reward function, it is also quite costly to do multiple iterations [66]. Therefore, the solution of running the system first in solution has been used. This can however not completely replace the real-time learning, as even small differences in the environment can result in totally different behaviours.

The ability of BC method to deal with high-dimensional inputs depends on the algorithm. Both DMP and ProMP are often not able to do so, whereas GMM, GPR and HMM are [39]. As RL often uses a continuous stream of data, the amount of data becomes large. This makes it often quite difficult for RL method to deal with robotic learning. Nevertheless, PS has proven to be able to scale well with high-dimensional state space [5]. The same cannot be said for value function methods.

The last criteria relate to *online* learning. For each method, online implementations could often be found. However, these implementations often requires some adaptation to the original method. Therefore, this criterium states whether methods were originally meant for online implementations. BC method usually obtained the demonstrations first, after which a model was computed. Recent can be found for online implementation of this method [23, 29], but as stated they require some adaptations of the original method. The principle of IRL is to use the demonstration to find a reward function, and use this to compute a policy (using some RL method) [59]. As the updating of the policy is done iteratively, this method can also be stated as an online learning process. RL method have always been used in an online manner. As the policy is iteratively updated based on the sensory feedback [66]. In contrast to the IL approaches, is offline usage of RL not common [71].

2.6. Summary

In this chapter, the main goal was to give an overview of existing method for robot teaching. In this section, the findings will be summarized.

- **IL and RL**: Two main approaches (besides the traditional direct programming approach) have been identified, which both have the benefit that the behaviour does not need to be manually programmed. IL uses demonstration, exerted by an expert, to learn skills. Which makes the learning easy and natural. However, due to these demonstrations, the skill of the robot is limited by the skill of the expert. In some cases, IL also has to deal with the correspondence problem, which is due to the different embodiment of the learning and the expert. Both of these limitations do not count for RL, as it does not require an expert teacher. However, RL is often more difficult to

specify, and choosing of an incorrect reward function can have tremendous results. In section 2.5, a comparison between these two methods have been given.

- **BC and IRL:** IL problems can be categorized into two main approaches. The BC approach tries to optimize the policy, by directly mapping. Most implementations are quite simple and computational efficient. However, often it only tends to work when large amount of data is required, due to a compounding error cause by covariant shift. This is not a problem when using IRL. In addition, IRL also, performance well in a suboptimal space, which is not the case for BC. This comes as the cost of computational efficient. A ~~comparison of these methods has been given in section 2.5.~~

- **Value function and PS:** The RL approach value function tries to find an optimal policy, by iteratively optimizing the value function. A disadvantage of this approach is that it often difficult problem in high-dimensional state and action space, which is problematic for robotic application. Therefore, PS can be a good alternative. This method directly learns the optimal policy. However, this method can be a bit more difficult to implement. A special type of PS method is AC. This method aims to combine the advantages of the value function and PS approach.

<div align="right">

3

</div>

Continuous Skill Learning

Chapter 2 focused on what methods can be used to teach a certain skill to a robot, either by imitation of an expert or by learning based on a (predefined) cost/reward function. In this chapter, the focus will be on how the usage of these methods changes the skill of a robot throughout the learning process. As this paper focusses on online learning methods, it is assumed that the behaviour which the robot will exploit at the start of the learning process, will not be the same as at the end of the learning process, i.e. it will incrementally become better.

The remaining of this chapter consist of an introduction to different learning outcomes in section 3.1. Next, the effect of skill learning, using different state inputs, will be examined in section 3.2. Besides learning a policy, using Imitation Learning (IL), Reinforcement Learning (RL) can also be used to learn policies. Therefore, section 3.3 is dedicated to method using a reward/cost function. Lastly, the findings will be summarized in section 3.4.

3.1. Learning Outcome

To learn a certain task, the learning outcome needs to be defined. In [9], three learning outcomes for IL problems were defined: a policy, a reward or cost, or a plan. The choosing of the learning outcome depends on the task and the associated constraints. The first two learning outcomes have already been discussed in chapter 2. The latter, however, requires some additional explanation. This method includes the learning at the highest level of task abstraction. Here it is assumed that the task is performed according to a structure plan, made up of several sub-tasks of primitive action. In reality, this method would include multiple policies, or rewards or costs. Therefore, this will not been seen as an individual aspect in the remaining of this chapter. Instead, if discussed, it will be categorized in one of the other types of learning outcome.

~~When researching the reward or cost function as a learning outcome, the literature for online learning have proven to be limited. Most implementation pre record all demonstration and determine the function, therefore offline [84]. Therefore, this will be excluded from further research.~~

Looking at policy as a learning outcome, Ravichandar et al. [9] defined different taxonomies to classify the different methods: policy input, policy output and policy class. Both policy input and policy output are quite straightforward. It should be noted, that the policy output is equal to the abstraction earlier discussed in section 2.2 (table 2.3). Policy class refers to the class of mathematical function to which the policy belongs. Using this as a classification would seem to be a bit arbitrary. It was chosen, when relating to policy, that comparing different inputs would seem the most logical choice. Reasons for this, is that whatever type of methods is used, the demonstrated data always comes into play. Which makes the content of the dataset of importance.

In the learning scheme of the policy, there are often two segments which make use of some state input: the policy (or controller) and the robot. For the classification of the different policy state inputs, as only online learning will be considered, the control scheme will either be complete close-loop or it will

close-loop with an additional feed-forward loop. In the latter one, the feed-forward loop will, in most cases, consist of one demonstration. This is often the case for approaches using Dynamic Movement Primitive (DMP). This means that the state input for the policy and robot will not be the same. In this section, the state input is classified as the state input used for the policy. Looking at this input, there are again multiple types. However, in this chapter, only state input will be examined.

3.2. State input

Different state inputs for the learning algorithm have been identified [9]. The different types of state input which will be examined in this section are position, joint angle, force and torque. While there are more state inputs to be found in literature, these are the ones most frequently used. An overview of the different papers which will be discussed in this section is given in table 3.1.

3.2.1. Position

Motion is often used for state input for policy learning using IL methods. This motion can consist of information of the positions and/or orientations of the robot, but also of its velocities and acceleration. There are multiple ways to require this type of data. One method is to use the embedded sensors in the robot. This is especially effective when using kinaesthetic teaching for the requirement of the data. However, often the data should be required from observing a (human) expert. In these cases, extracting the information can be done using ~~either one of two other frequently used approaches:~~ vision and motion sensors [49, 87]. Most state-of-the-art approaches consider vision, as it seems to be the most natural way to capture information from the human. However, these systems must deal with problems as occlusion, appearance change and complex human-robot kinematic mapping. Therefore, using motion sensor instead maybe a more practical approach. This allows for the tracking of the motion preciser and a simple mapping can be established.

A framework for supervised teleoperation with online learning, was proposed in [85]. By using an online Bayesian non-parametric learning algorithm, motions can be, online, represented as task-parametrized Hidden Semi-Markov Model (TP-HSMM). These motion can be executed autonomously using a model predictive control (MPC) approach. In the paper, the principle is illustrated using a simplistic example in a planar environment, as visible in fig. 3.1. By means of teleoperation, ~~where the mouse cursor is used,~~ three demonstration were exerted. This resulted in a model with six Gaussian components. Once it is the remotes' robot's turn, it is visible the system is able to create a path to the goal from a new initial position.It should be noted that the starting point is ~~here~~ indicated by the demonstrator. A graphical representation of this idea has been presented in fig. 3.2.

While the planar example gives a clear overview of the main principle of this approach, the eventual goal of the paper [85] was to execute a peg-in-hole (PiH) task. In this experiment, the left arm of a robot is used to kinaesthetically demonstrate the behaviour, and the right arm to remotely reproduce it. Here the model is update online using five trials. Due to the updating, the ~~demonstrated data does not need to be remembered.~~ The end goal was found using a tracker. The reproduction phase showed

Table 3.1: Overview of references using different types of state input. For the method, the encoding and learning method is meant. The demonstration and task which are named in this table represent the relevant information for this chapter. In some cases, other demonstration types and/or tasks were also executed.

| Author & year	Ref.	State input	Method	Demonstration	Task
Havoutis et al. (2017)	[85]	Position	TP-HSMM	Kinaesthetic teaching	PiH
Calinon et al. (2007)	[37]	Joint Angle	GMM + GMR	Shadowing & Kinaesthetic teaching	Gesture imitation
Pastor et al. (2011)	[80]	Force	DMP	Kinaesthetic teaching	Grasping
Nemec et al. (2013)	[29]	Force	DMP + ILWR	Teleoperation	PiH
Peternel et al. (2017)	[23]	Force	DMP + LWR	Shadowing	Sawing
Peternel et al. (2016)	[86]	Torque	DMP + LWR	Shadowing	Periodic movement.
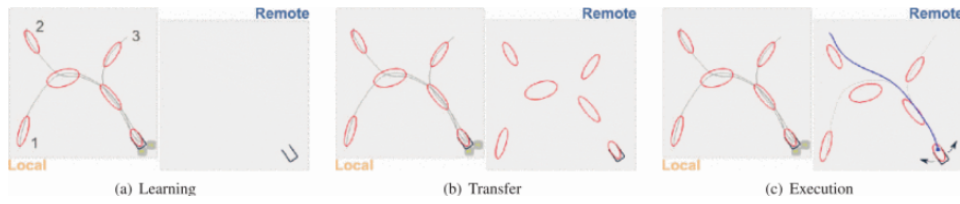
Figure 3.1: Planar example of learning and control with the TP-HSMM [85]. Here, in each plot, left represents the local system (operator's side) and right the remote system (robot side). (left) represents three motions and six corresponding Gaussian (here Gaussian are added after each demonstration). (middle) shows the transfer from the local system to the remote system. (right) shows a new motion which is generated by the MPC controller. It should be noted that this can only be created once the operator has initialized a new initial state.
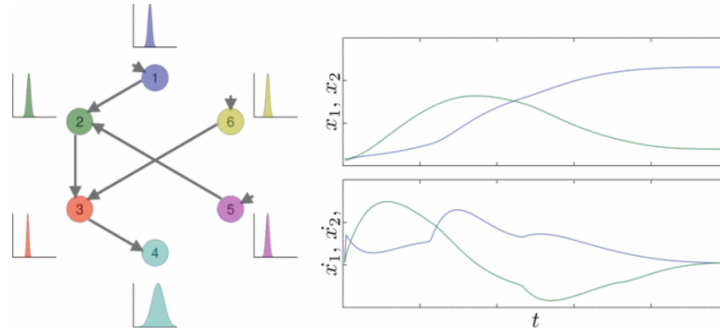

Figure 3.2: Planar example with (left) the graphical representation of the transition matrix and duration probabilities for each state, where the three paths are formed according to the three demonstrations. And (right) the executed motion on the remote side starting from a new position, using TP-HSMM and the MPC controller.

that the model generalized well, even when the starting and/or goal position was much different from demonstrated data. In [88] this framework was expanded to disambiguate differences between the operator's and the robot's workspaces.

3.2.2. Joint angles

To calculate the position of the end-effector, often the different joint angles are used, ~~which can be translated into a position~~. Comparing position and joint angles as state input, joint angles often allow for less variability in the configuration of the robot. Where position often only defines the position of the end-effector, does the joint angles define the entire configuration of the robot. In case there is only one joint, there is only one possibility of the configuration of the robot using position as control input. However, using multiple joints allow for many more possible configurations. Examining this trough time the difference could be that using the joint angles would allow for only on trajectory of the angles whereas in case of position still multiple solutions are available. ~~Therefore, using more joint reduces the solution space. While this would result in a more stable system, this also has the disadvantage that it is less able to adapt to changes in the environment and is thus less save. Another main problem arising with the usage of joint angles as state input is the correspondence problem, as the joints of an expert might not be the same as those of the robot. However, this can be avoided by using kinaesthetic teaching for demonstration.~~

In [37], a wearable device was attached to the human expert, which included 8 sensors which each provided its absolute orientation. The goal of this research was to transfer a motion from a human to a (small) robot. This consisted of two steps: an observation step and a kinaesthetic teaching step. In this first step, the recorded joint angles are projected in the latent space and encoded using Gaussian Mixture Model (GMM). The training of the parameters is here done using Expectation Maximization (EM). In the second step, kinaesthetic teaching is used to refine the motion of the robot. The updating of the GMM is done by estimating online the new incoming data and the previous estimation of the GMM parameters. One of the approaches used for updating the GMM is the direct update approach, which uses an adapted version of EM. Results for a specific gesture are shown in fig. 3.3. The usage of both the motion sensor and kinaesthetic teaching allows for dealing with the correspondence problem.
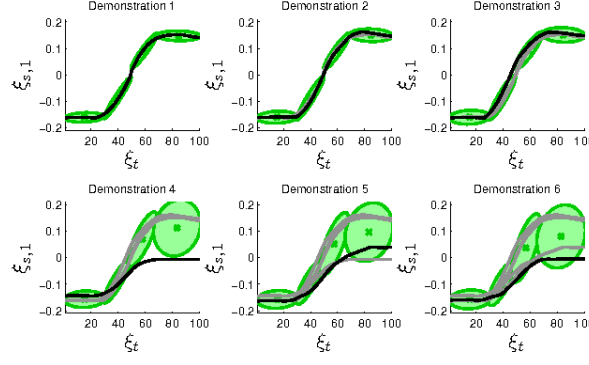
Figure 3.3: Updating of the GMM using the direct update incremental learning approach [37]. The graph shows the encoding of the data in GMM. Each algorithm uses only the latest observed trajectory (illustrated by a black line) and the previous estimation of the GMM parameters, to update the models. Demonstration 1-3 made use of only the motion sensors, and in demonstration 4-6 the additional information was required using kinaesthetic teaching.

Besides the direct updating method, another method was used to update the model: the generative method. Both these methods were compared to an offline learning approach: batch learning. The results for three different gestures are visualized in fig. 3.4. For the reproduction of the motion Gaussian Mixture Regression (GMR) was used. The results show that each method was able to capture the essential characteristic of the demonstrated data.
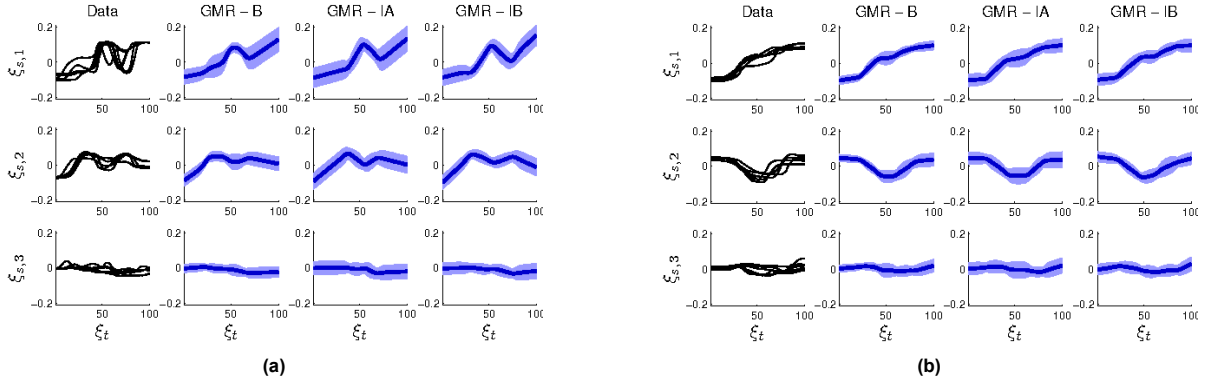

Figure 3.4: Reproduction of different gesture movements using GMR in a 3D latent space of motion [37]. *(a & b)* visualize three different gestures. For each gesture three models are trained: one with different batch *B* and two incremental training methods *IA* and *IB* representing the direct update and generative method.

3.2.3. Force

While position and joint angles can provide a suitable solution in some cases, often they do not account well for adaptation in the environment. The reason for this is that they are simply unaware of the changes. ~~This could be fixed using sensory information describing the environment, e.g. vision.~~ Take for example the grasping of a cup, if the cup is not located at the same position as it has always been and there is no additional feedback, then the robot will simply not grasp the cup. A solution for this is the usage of force as a state input. Here the robot could feel whether it is touching the cup and adjust its trajectory based on this.

An example where force input was used is [80]. Pastor et al. used a DMP framework to encode stereo-typical movement and extended this for online movement adaptation using sensory feedback. ~~In specific,~~ this framework was applied to a grasping task, using DMPs sensory feedback and nonlinear position and force control. The learning scheme used for this is visualized in fig. 3.5. The low-level controller was separated for the end effector (i.e. the hand) and the for the fingers. As visible in the graph, the used DMP is changed as a result of the demonstrated data and the previous task execution. In the experiments, the proposed control mode was compared to an open loop and a force control without DMP adaptation control mode. The task for the robot was to grasp a certain object from a table,
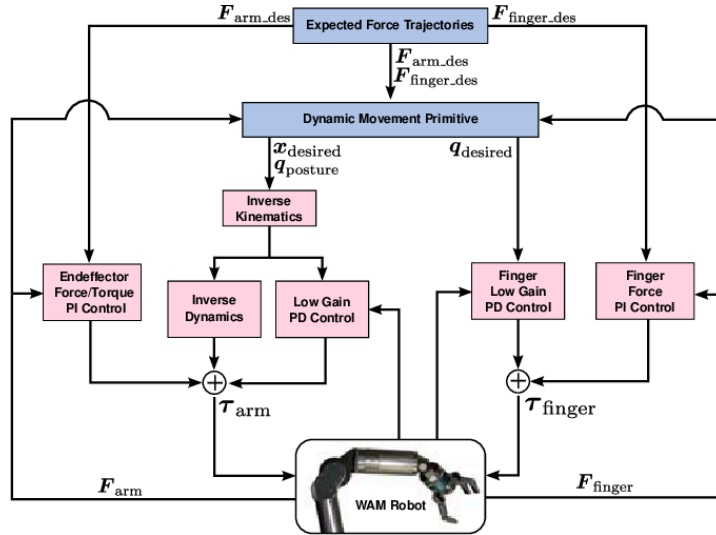
Figure 3.5: Learning scheme for a grasping task Pastor et al. The low-level controller for the end-effector (i.e. hand) and fingers are separated. The control law for the finger τ_{arm} is a combination of a position controller (consisting of a velocity-based operational space controller and an inverse dynamics law and feedback error compensation in joint space) and an end-effector force controller. The control of the fingers τ_{finger} consist of a position PD controller and a force PI controller.
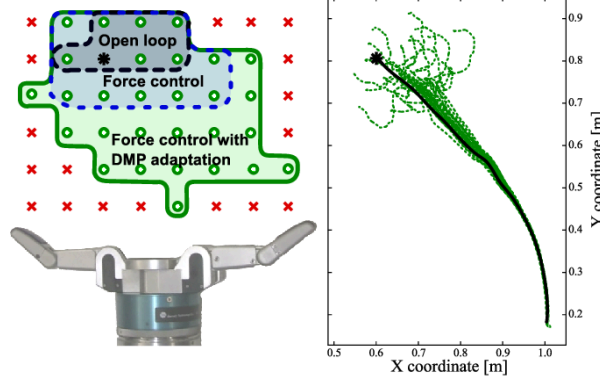

Figure 3.6: Result of grasping attempts after misplacing a flash-light [80]. (left) shows at which positions, the different control modes are still able to grasp the flash-light and (right) shows in black the open loop trajectory and in dashed-green the adapted trajectories after touching the object.

in the case of the experiment a flash-light. For each control mode, it was tested how they would adapt if the flash-light would be displaced. The results are shown in fig. 3.6. A visible on the left, using DMP with adaptation, significantly increases the area where the robot is still able to grasp the flash-light. In the provided video, it was visible that the robot would always try to go to the position where it "thinks" the flash-light is located. However, if a force input is provided which indicates that the flash-light is not at the position where it is assumed to be, then it adapts its behaviour according to this force feedback. Therefore, it could be concluded that the reason that the robot is not able to grasp the flash-light at any position in the area (as indicated with the red crosses in fig. 3.6), even when the adaptation is used, has something to do with not being in contact with the flash-light at these points (either due to going over it or not reaching it). To deal with this, another form of feedback could also be used, which does not require touching the obstacle. A good example for this is vision, based on which the position of the object could be extracted.

Nemec et al. [29] presented a policy learning and adaptation algorithm, using DMP, for a PiH task. Its learning scheme has been presented in fig. 3.7. The main difference, compared to other IL approaches, is that not only trajectories, but also forces and torques arose during the task demonstration. To test this scheme, two types of robots were used. However, as both obtained similar results, only one will be discussed. This robot, a 6 Degrees of Freedom (DoF) UR5 equipped with gripper and a force/torque

wrist sensor, uses a high gain non-compliant controller and therefore allows for the implementation of an admittance force control law. The robot is also equipped with a magnetic tracker to track the peg's pose and orientation. The demonstration was exerted using teleoperation with a pose tracker. To test the proposed algorithm, three PiH tasks were done, of which the results shown in fig. 3.8 are of the second task where the robot has to insert a square peg when the baseplate has been rotated and translated. As expected, iteration result in a lower offset of the position. However, as visible, the offset is not completely removed. This does not necessarily have to be an issue, as multiple trajectories can be used to exert a similar task. Looking at the force, it is visible that the force used in the learned trajectories is lower than the demonstrated one. The paper names as reason an imperfect force sensor calibration. This is most likely to be accurate, as the goal is to minimize the error between the demonstrated and generated forces. Therefore, directly trying to recreate the demonstration thus lead to suboptimal performance. It was also noticed that the robot would try to replicate the learned force/torques rather than the position/orientation of the human demonstration. This, however, does not necessarily have to be a problem, as there can be multiple trajectories which result in the execution of the task.
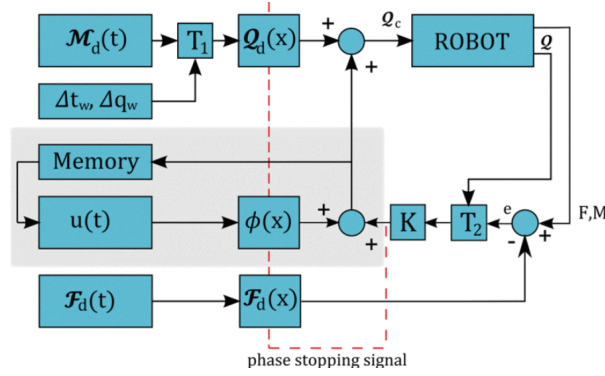

Figure 3.7: Learning scheme of PiH tasks, describing an iteratively updated DMP using both position/orientation and force/torque input, where $\mathcal{M}_d(t)$ denoted the original demonstrated trajectory encoded by DMPs, Q_t the output signal obtained by integrating the transformation equations of the DMPs and applying workpiece displacement ($\Delta t_w, \Delta q_w$) as estimated by a vision sensor (this takes place is block T_1), $\mathcal{F}_d$ denotes the captured forces and torques captured during demonstration they are encoded using a linear combination of Radial Basis Function (RBF), the eventual goal is to minimize the difference between the measured force/torque and the desired force/torque (denoted as e), in T_2 this error is transformed into robot base coordinates. By using admittance control, the position/orientation offset is calculated and added to the existing offset of the previous iteration step $\phi(x)$. The motor commands Q_c are then a combination of the motor commands obtained from the DMPs (Q_t) and the offset. The iteration procedure is repeated until the measured forces, matches the desired forces.
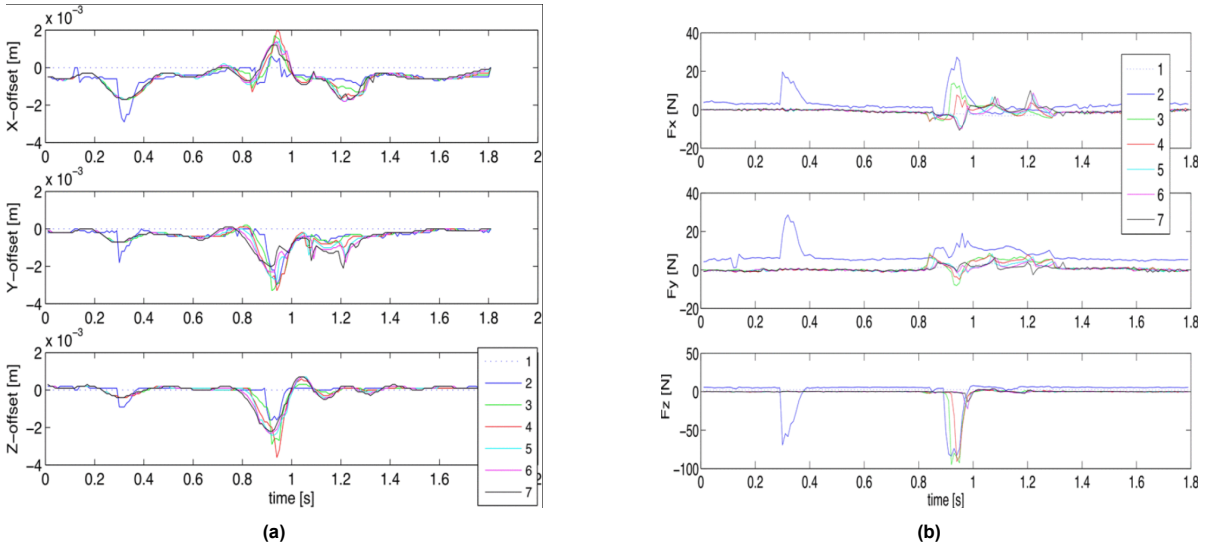

Figure 3.8: Results of PiH task with a rotated and translated baseplate [29]. *(a)* The learned positional offset of 6 consecutive cycles and demonstrated trajectory denoted by 1, which has no offset. *(b)* The force measured during the execution of the task in 6 consecutive cycles, again 1 denoting the demonstrated trajectory.

All experiments till now have shown the ability of a robot to learn from demonstrations exerted by a human expert. Peternel and Ajoudani expanded this idea by letting robots learn from each other [23]. They presented a method where one robot was assumed to be an expert. Before becoming an expert, this robot had to require its skill from a human expert. This concept is presented in fig. 3.9. The interest of the paper [23] lied in the area illustrated with red. This part consisted of three stages: a learning stage where the novice robot is completely compliant, and the expert robot has a high stiffness, a stage where the novice robot learns the desired impedance behaviour which encodes a follow/leader role in different phases of the task, and a third stage where both robots are assumed to be an expert. DMP was used to encode the trajectories in the first stage, and Locally Weighted Regression (LWR) to learn these encoded trajectories. In fig. 3.10 the learning process of the novice robot is shown. The interesting part occurs in the third stage. In theory, the stiffness pattern of both robots should be the same, except from a phase shift of half the period time. However, as visible, the stiffness pattern of the novice robot is not at all the same as the one of the expert robot. In previous shown experiment [29, 37, 80, 89], the aim was always to mimic the human expert as good as possible. However, as this experiment uses a collaboration between the robots to learn the skill, a randomness could occur. A theory could be that the has, due to this randomness, improved the skill in comparison to the expert robot. This could therefore be an interesting topic of research. However, with randomness, the possibility could also be that behaviour becomes worse.
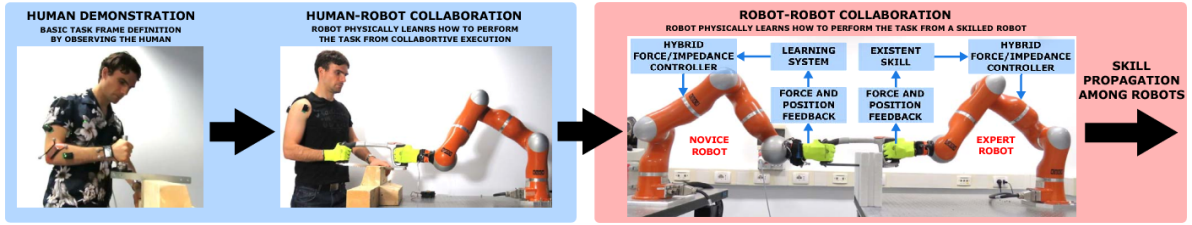

Figure 3.9: The skill learning of a novice robot [23]. The blue area illustrates the skill acquisition for an expert robot, based on the concept presented in [90]. This start by demonstrations from the human which is observed by the robot, this allows for defining the task frame control framework. Next, the expert robot uses this knowledge to collaborate with the human and learn the task specific, using physical interaction. The next area (indicated with red) illustrates the proposed method in the paper, where the novice robot learns from the expert robot. This part can be propagated among other robots.
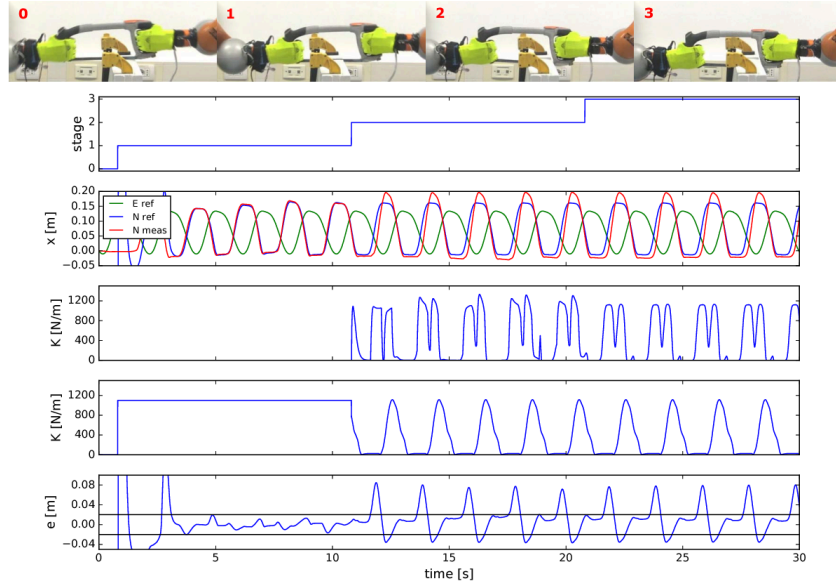

Figure 3.10: Results of robot-robot collaboration experiment on two-person sawing task [23]. The sequence of photos in top shows the sawing progress throughout different stages. The first graph shows learning process stages as described in Section II-C. The second graph shows the sawing motion in x-axis. Thegreen line represents the reference motion of the expert robot, the blue line represents the reference motion of the novice robot and the red line representsthe actual motion. The third graph shows the stiffness of the novice robot. The fourth graph shows stiffness of the expert robot. The fifth graph shows theerror between the reference and actual motion used in the stiffness learning process. The thresholdsethare drawn by black lines

3.2.4. Torques

A last frequently used state input is torque. ~~The usage of torque as a state input can be especially useful for periodic motions.~~ In [86] an example is given where an exoskeleton is used to reduce the effort exerted by an expert. While this is often measured using electromyography (EMG), in essence the goal is to reduce the torque exerted by the human. When using positional signals for this problem, the robot would as output use a learned torque trajectory. This would result in the human having to produce an opposing torque to change the joint angle. Therefore, instead, it was found to be more useful to use the torque (measured using EMG) as an input signal.

In [86] the torque was used to learn a pressure trajectory. The presented control scheme is graphically shown in fig. 3.11. The sensory feedback, resulting from the human, was measured using EMG. The system used DMP to encode the trajectory and LWR to learn it. In the paper, different experiments were conducted, each having the goal to reduce the human muscular effort as much as possible. In the tasks, the human subject had to move an object repetitively from one position to another at a certain frequency. While doing so, the exoskeleton would try to adapt to the human and, by doing so, reducing the effort. One of the experiments used an elbow exoskeleton, meaning that the subject would only be supported by torque exerted on this specific joint. The subject had to execute three different phases, where each phase consisted of a target elbow joint range and frequency. The results are visible in fig. 3.12. The third plot shows the muscle activity. As visible, each phase first consist of the subject putting in much effort until it reaches the desired behaviour. Once it researches this behaviour, the effort is reduced to zero. To do so, assistive torque is used. As visible in the second plot, it would take a bit of time to adapt to a new behaviour.
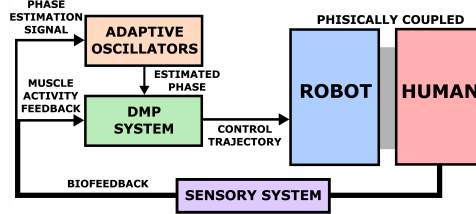

Figure 3.11: Control scheme of an exoskeleton as presented in [86]. The human is included in the control loop, by using sensory biofeedback obtained with EMG. The goal is to minimize the human muscle activity feedback by learning an appropriate assistive torque behaviour in the joint. The adaptive oscillator extract the phase and frequency information from the human biofeedback and adjust the phase and frequency of the learnt feed-forward trajectories based on this. Based on this estimated phase and the muscle activity feedback, a DMP is created, which results in a control trajectory.
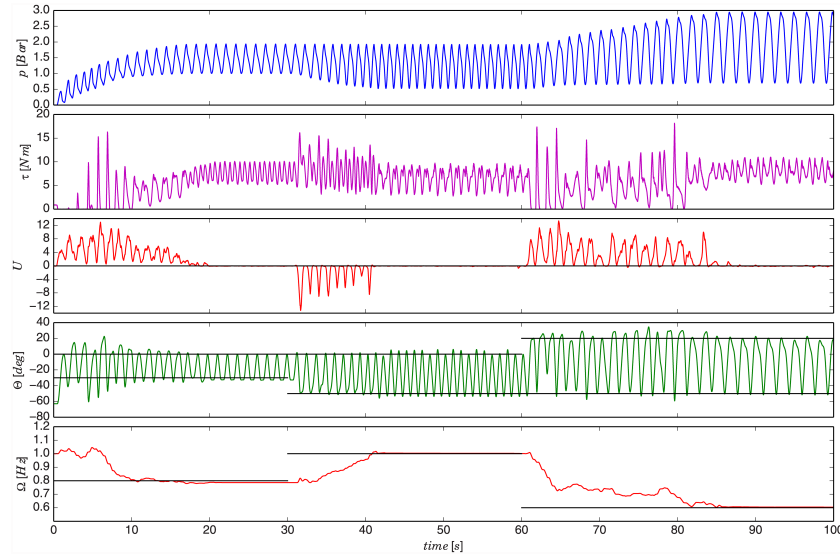
Figure 3.12: Results of experiment on elbow exoskeleton using a trained subject [86]. From top to bottom the different plot illustrate the: command PAM pressure learned by the DMP system p, the joint torque calculated based on the force measurement of the load cell τ, the human muscle activity based on feedback obtained with the measured EMG U, the angle of the robot elbow joint Θ and the frequency of the motion as extracted by the adaptive oscillator from the muscle activity feedback Ω. The reference states are indicated by a black line.

3.3. Reward or Cost

RL has the advanced that the learned skill is not limited by the knowledge of the demonstrator. However, it has the limitation that it requires the reward or cost function to be known. The choosing of this function is critical as the resulting behaviour depend on this. If this function is not known, using Inverse Reinforcement Learning (IRL) could be an option. ~~However,~~ this again requires human demonstration, thus adding the drawback of being limited by its skill. In this section, the continuous learning of a robot using a reward or cost function is shown. The skill improvement will be analysed based on the improvement in the received reward or reduced cost. In the first part, section 3.3.1, this will be done using a regular RL approach. In addition, an implementation of RL using human advice is described in section 3.3.2.

3.3.1. Regular learning

In [91], actor-critic RL (described in section 2.3.4) was implemented to execute a trajectory following task. In the paper, two control methods were introduced: a RL-based reference compensation and a RL-based input compensation. The former is formulated as a function of the joint space error and its derivative. The latter directly compensates the reference signal fed to the system. Two of the tracking task, used to test both RL approaches, will be discussed below. For these tasks, a UR5 robot, controlled using velocity commands, is used.

The first task is to follow a discontinuous trajectory, namely a square. The goal is here to reduce the error in the z-axis. The video provided by the paper shows the learning process of the robot. The first iterations show that the robot is a bit higher above the table than required. It also shows an overshoot when trying to follow the squared shape trajectory. It is noticeable that when the square shaped trajectory start, there is nearly no delay in trying to follow it. During the learning process, the overshoot becomes incrementally better. The results also show that the skill improvement is the highest at the start of the learning process, and slowly convergence until the average best performance is obtained. The learning process based on the sum of rewards (return) is visualized in fig. 3.13. For the input based method, the return of each agent (joint), slowly converging to an average best solution. In contrast, the reference based method shows an erratic behaviour. The paper stated two possible reasons for this; Firstly, it can be caused by the RBFs which are initialized with an inappropriate value, causing the learning curve to deteriorate before improving. And secondly, that the reference discontinuities introduce very large errors, which cause the policy value function to first behave erratically before converging to a stable behaviour.

The second task is to follow a continuous trajectory, namely a circle. The goal is to minimize both the x-axis and y-axis error. The provided video shows that initially the robot is already able to follow the circle trajectory quite well. Looking at the two different methods, fig. 3.14, it is visible that using the input based compensation result in much faster convergence. This is a result of the directly reducing of the trajectory error, which means it directly compensates for the joint velocity. The response of this
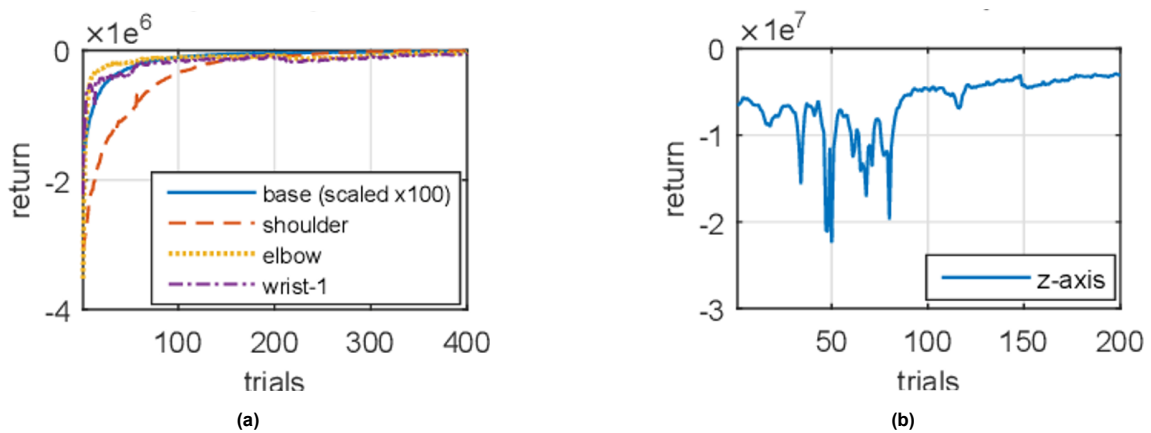

(a)

(b)

Figure 3.13: The sum of reward (return) for the squared trajectory following [91]. With *(a)* the RL-based input compensation, and *(b)* the RL-based reference compensation.
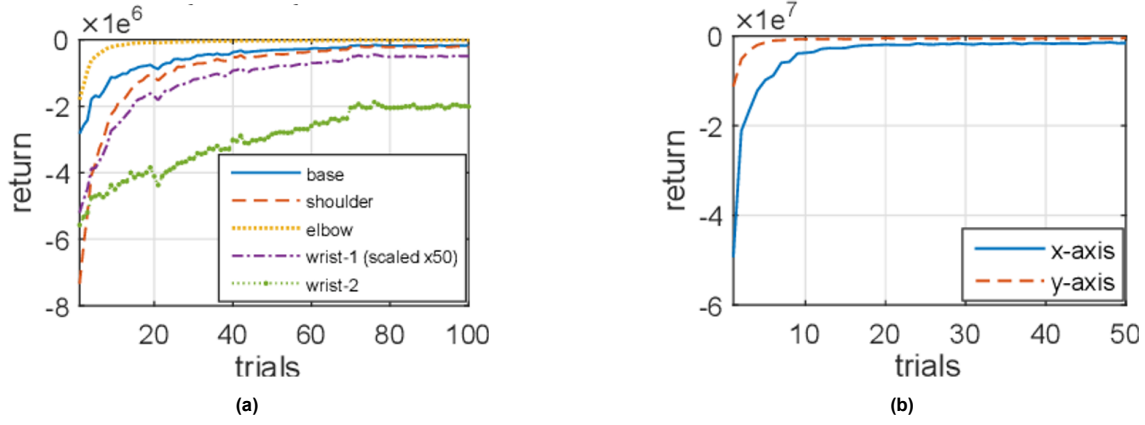
Figure 3.14: The sum of reward (return) for the circle trajectory following [91]. With *(a)* the RL-based input compensation, and *(b)* the RL-based reference compensation.

method is however a bit slower, as the corrected trajectory is tracked using a nominal PD controller. Lastly, as this method modifies the reference instead of the control input, the result is slightly smoother than that of the input based controller.

In [91], multiple reasons are stated why either one of the control functions would be better for certain criteria (response time, convergence speed, smoothness, etc.). Besides these criteria, the importance of choosing a suitable control function for specific task, can be seen. Where the reference-based compensation does not even work for the following of the discontinuous trajectory, results it in the better behaviour for the continuous trajectory.

3.3.2. Human Advice

In section 2.4, the usefulness of combining IL and RL was already expanded on. This idea was adapted by Celemin et al. [92]. Here, human feedback was used to excel the policy beyond the capabilities of a human expert. In their research, two methods using feedback were used; Firstly, an adapted version of Corrective Advice Communicated by Humans (COACH) was used. This method can be used to incorporate the teacher correction in the policy search (PS) loop. During execution, the human teacher can give occasionally advises, to correct the behaviour, during execution time. Secondly, they introduced *interactive PS*. This method, also depended on COACH, used human guidance to influence and bias the exploration of the PS algorithm. This was done in the from of exploration noise.

To examine the performance of the approaches, a *ball-in-cup* task was used, as shown in fig. 3.15. Here the reward function was separated into two parts: one where the height of the ball is lower than the cup and one where it is higher. This was an extension of the earlier used approach [93], as that reward function would not provide any useful information if the ball was lower than the cup. Due to the extension, the task can be considered as two sub-tasks: *swinging the ball* and *catching the ball*. The policy was represented using Probabilistic Movement Primitive (ProMP) and PI^2 was the base for the PS algorithm.

The first demonstration used a policy derived using kinaesthetic teaching as an initial starting point. The learning curve of this task has been presented in fig. 3.16. It is visible that both method using human feedback converge initially much faster than the standard PS method. COACH and the interactive PS are able to achieve policies which can catch the ball at 10 and 15 episodes, respectively. Whereas the original method has this at 60. Noticeable is that the COACH implementation does not improve much after the initial catch. This can be explained as the human will stop giving advice once the ball has be caught, either because the need for improvement is not evident or it is not necessary. This shows the inability of human to improve once they are near an optimal policy. Controversy, the interactive PS learns a bit slower at the start, which is probably due to some earliest roll-outs in the update process. However, it keeps improving until it researches the best average performance around 70 episodes. The pure PS, might converge slower, but reaches, just as the interactive PS method, a better performance than the solely COACH method. These results show that the advantages of combining IL with RL is that
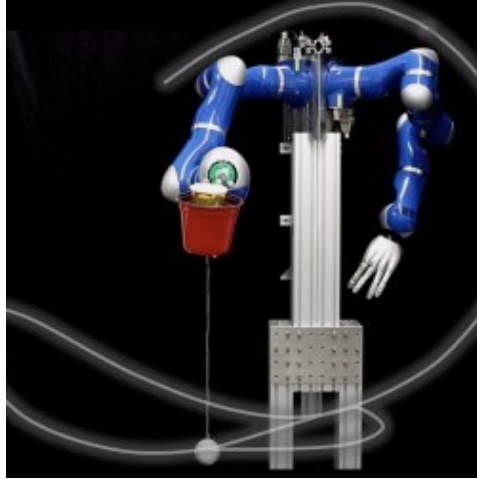
Figure 3.15: Snapshot of ball-in-cup experiment [92].

it also combines its features: faster convergence (due to IL) and researching of a better performance (due to RL).

In the second set of experiment, the relevance of choosing a good initial policy is shown. In contrast, to the earlier experiment, does this one start from scratch. The results are visualized in fig. 3.17. The effect of starting from scratch is visible as the starting point is now at 1 instead of 0.1. This has especially a big effect on the PS method, which takes a lot longer to converge now. Showing its sensitivity to the initial policy. At the start the learning process is quite slow as the random movement seems to reduce the effect of the previous ones. The results do not even show the PS receiving a behaviour where it is able to catch the ball. However, if more episodes were conducted, it is reasonable to assume that it would eventually be able to. Looking at the interactive PS and the COACH method. A similar convergence behaviour is visible as in fig. 3.16. While it takes them both a bit longer to catch the ball, they eventually achieve similar performance as before.

This research shows two important factors. Firstly, the effect of combining IL and RL on the learning time of the policy. And secondly, the usefulness of choosing a good initial policy.
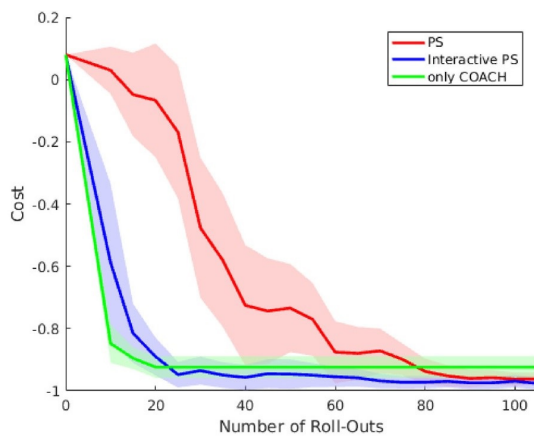

Figure 3.16: Convergence curve of *ball-in-cup* when learning from an initial demonstration [92]. This task was learned, using a standard PS method, an interactive PS method, and PS with COACH.
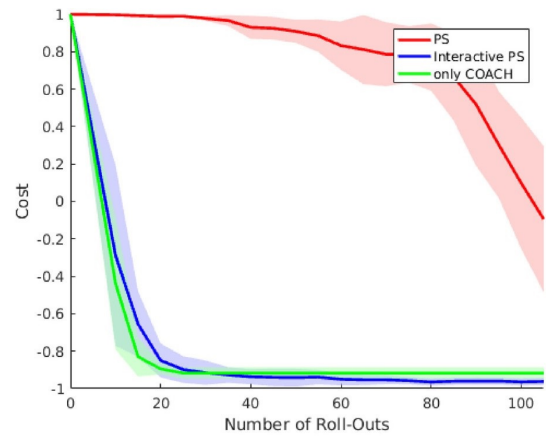

Figure 3.17: Convergence curve of *ball-in-cup* when learning from scratch [92]. This task was learned, using a standard PS method, an interactive PS method, and PS with COACH.

3.4. Summary

The goal of this chapter was to illustrate how skill can change during the learning process. In this section, the main findings will be summarized.

- **Learning outcome:** There are multiple types of learning outcomes, the two elaborated on are: a policy, and a reward or cost function. For online learning, the research for the latter has proven to be limited. Policy learning can be classified based on the type of state input used. The four examined in this chapter are: position, joint angle, force and torque.

- **Position and joint angles as state input:** The benefits of position and joint angle often are quite similar. This is logical, as the position is often a translation of joint angle. While both these inputs have proven to result in good performances, they also seem to be limited when relating to adapting to environments. Changes in the environment cannot be easily adapted to by just using the configuration of the robot. Instead, some types of additional feedback needs to be provided in these cases: for example, the position of an object based on vision.

- **Force and torque as state input:** An advantage of using either one of these state inputs, are their ability to adapt to the environment. ~~Especially in situations where the task differers slightly from the demonstrations, using these inputs works quite well.~~ When slightly shifting an object from position, if the robot still touches it, it would be able to successfully grasp the object. However, just as for position and joint angles, additional information is required to find poses totally different from the original.

- **Collaborative task:** These types of task are promising for future research. Task using some kind of demonstration (teleoperation, kinaesthetic teaching, etc.) try to find a behaviour which mimics the behaviour of the expert as good as possible. Whereas this is also the original goal of the collaborative task, research shows mutated behaviour which can - but does not necessarily have to be - better than those of the expert.

- ~~**Control system:** the usage of two different control methods in [91] stretches the importance of choosing a suitable function for a certain task.~~

- **Human feedback**: The results in [92], show the benefit of using human feedback in addition to RL. The main advantage, is the improvement in convergence time, while as a result not losing the final performance.

4

Discussion

This literature report has aimed to give an overview of methods which could be used for continuous skill learning. In this chapter, the main finding will be discussed. The main goal is to identify the bridge between the different methods identified in chapter 2 and the effect of skill learning in chapter 3.

To start, many methods have been identified which can enable continuous skill learning. The two most broad being Imitation Learning (IL) and Reinforcement Learning (RL). Both come with their advantage and disadvantage. IL allows for a reliable and save design, which often allows for good generalization purpose. However, a major drawback is the fact that they are limited by the skill of the expert. In literature, three solutions who show potential to tackle this problem has been identified.

The first solution, requires an extension of the process of skill learning by the usage of an additional step. A basic skill could be required by observing an expert, which is the general principle of IL. After learning this basic skill, the behaviour could be improved using a collaborative task [23]. This collaboration allows for the learning robot to be less constrained and thus provides the opportunity to research the solution-space further than the one described by the expert.

Another option would be to allow for a lot of noise in the data [29]. In essence, this results in more randomness in the solution and thus has the potential of becoming better. However, a major drawback of this idea is that it also has the potential of becoming much worse. This method can also not provide any stabilities guarantees, which potentially results in an unsafe system.

A last promising solution, is to make RL complementary to IL. Here, IL would be used to learn the demonstrated skill fast. After which the skill can be improved using RL. While this method seems to be promising, research on this topic seems to be limited. Reasons for this could be that the both the field of RL and IL are relatively new, making this combined topic something for the future. However, it can also be speculated that combining these two methods can be somewhat challenging.

The idea behind each of this method is that a larger part of the solution space will be examined to obtain a good behaviour. This property is typically embedded in RL the method. This method is, however, limited by the choice of reward function. While most research mentions the importance of choosing a correct reward function, the effect of choosing it incorrectly seems to be limited. Instead, most implementation have either based their function on previous research or they neglected to mention why this would be the best function in comparison to any other possibility. Even though this function often does not seem to be having the best reasoning behind it, it often still provides a solution for the given problems. This results in the question of what a good solution is. Some research have used the completion of a task as a reward function. However, this binary approach often does not seem to be the best option, as not finishing of a task, even if you are close, is always seemed as the same failure. Therefore, reward shaping is necessary. This is a method for providing more frequent feedback on the appropriate behaviours. A method for finding rewards is Inverse Reinforcement Learning (IRL) showing to close link between RL and IL methods. This again results in the limitation of Behavioural Cloning (BC) to only be as good as the expert.

Even though Dynamic Movement Primitive (DMP) is not specifically designed to be an online learning method, it is the one most frequently used for these types of problems. Features which makes DMP favourable are its ability to efficiently learn and generate movements and its guarantee of goal convergence. Looking at DMP using the "core" version limit itself to one demonstration, which, if not optimal, result in a not optimal solution. This is shown in [29]. The ability of DMPs to update to new trajectories has been seen in [86]. Here, each time the human exerted a new periodic movement, this movement can be seen as a new demonstration, thus showing the ability of DMP to update.

Looking at the learning process, some form of feedback of the robot is required to improve its behaviour. While force feedback seemed promising to adapt to the smaller changes in the environment [80], it also showed that it did not account for larger changes. Therefore, the literature has proven that using sensory feedback alone is not enough. Therefore, some other feedback is also required, e.g. vision. While research can also be found relating to this topic, it has exerted the research in this literature.

5

Conclusion

This chapter aims to give a conclusion, based on the research question defined in chapter 1. This is followed by some advice for further work.

5.1. Conclusion

The research question on which will be concluded in this chapter is as following:

> How do methods, found in literature, enable continuous skill learning of robots, and how does the learned skill change during the learning process?

In the literature, two broadly applied approaches have been identified (chapter 3). Both allow for continuous skill learning of robots:

- **Imitation Learning (IL)**: learning a policy by mimicking some demonstrated task.
- **Reinforcement Learning (RL)**: learning a policy using a trial-and-error approach, where the behaviour is either rewarded using a reward-function or penalized using a cost-function.

Different Imitation Learning (IL) method allows for different learning processes. The main distinction can be made between methods using one demonstration and methods using multiple. For the one demonstration, the learner can either learn a policy in one go, which cannot be seen as an online implementation, or it can be used to compare the behaviour of the robot to. In this last approach, the policy is iteratively updated until the difference between the demonstrated data and the behaviour of the robot is minimized. This thus assumes that the optimal behaviour is not learned in one go. For multiple demonstrations, the learner can either obtain all demonstrations at once and compute a distribution based on this, which is again not an online implementation, or each demonstration is used to iteratively update the policy. The latter then takes each new iteration not all previous ones into consideration, but instead only uses the last generated policy.

RL in most cases enable online learning as each iteration the policy is updated. The skill learning shows that each iteration the behaviour becomes slightly better. Often the system starts with updating into one direction in the solution space (the one direction meaning towards an optimal/good behaviour), unit ill it overshoots. After the overshoot, the direction is shifted, resulting in a zigzag behaviour around the optimal solution. An example is throwing a ball in a net, where first the ball does not reach the net, which is followed by throwing over the net, and again throwing it in front of the net, until it eventually goes through it.

As solution IL method provide a reliable result which is evenly good as that of the expert. Whereas, RL is less reliable but has the potential of finding better solutions than IL.

5.2. Future research

During this literature study, multiple gaps in research has come to light. Therefore, in this section, some future research topics will be presented. These topics have been identified based on the papers research in this report and are therefore assumed not to be existent at this moment.

Staring, research on both IL and RL has been widely applied. There is, however, a lack of research comparing both methods. To do so, suitable metrics should be used. In most cases, these metrics will be quite similar to the variables on which the reward or cost function of the RL method depends upon.

While reward functions are often mentioned as being highly critical for the result of the RL algorithm, comparing different function for robotic learning has not been found. Therefore, it could be useful to compare different reward functions for the same task.

Literature has shown the potential of exerting an experts' skill by the usage of collaborative task for skill learning. As the demonstrated data is not directly transferred from the expert to the novice robot, there is potential of becoming better. This idea is of course still limited by the conditions of the task. However, applying this principle multiple times could result in a divergence of the original execution of a certain task. This again can also have positive results on the generalizability of the system.

Combining the features of IL and RL methods into one approach could result in the robot to exert the behaviour of using either of these approaches singular. This could therefore form an interesting research topic.

References

[1] Aude G. Billard, Sylvain Calinon, and Rüdiger Dillmann. "Learning from humans". In: *Springer Handbook of Robotics* (Jan. 2016), pp. 1995–2014. DOI: `10.1007/978-3-319-32552-1_74`.

[2] Luka Peternel, Erhan Oztop, and Jan Babic. "A Shared Control Method for Online Human-in-the-Loop Robot Learning Based on Locally Weighted Regression". In: IEEE, 2016, pp. 3900–3906. ISBN: 9781509037612.

[3] Duy Nguyen-Tuong and Jan Peters. "Model learning for robot control: a survey". In: *Cognitive processing* 12.4 (2011), pp. 319–340.

[4] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. "Reinforcement learning in robotics: Applications and real-world challenges". In: *Robotics* 2.3 (2013), pp. 122–148.

[5] Oliver Kroemer, Scott Niekum, and George Konidaris. "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms". In: *Journal of Machine Learning Research* 22 (2021), pp. 1–82. URL: `http://jmlr.org/papers/v22/19-804.html.`.

[6] Kyriacos Shiarlis, Joao Messias, and Shimon Whiteson. "Inverse Reinforcement Learning from Failure". In: 2016, pp. 1060–1068. URL: `www.ifaamas.org`.

[7] Andrew Y Ng, Stuart J Russell, et al. "Algorithms for inverse reinforcement learning." In: *Icml*. Vol. 1. 2000, p. 2.

[8] Zuyuan Zhu and Huosheng Hu. "Robot Learning from Demonstration in Robotic Assembly: A Survey". In: *Robotics* 7 (2 Apr. 2018), p. 17. ISSN: 22186581. DOI: `10.3390/ROBOTICS7020017`. URL: `https://www.mdpi.com/2218-6581/7/2/17/htm%20https://www.mdpi.com/2218-6581/7/2/17.`

[9] Harish C. Ravichandar et al. "Recent Advances in Robot Learning from Demonstration". In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (1 May 2020), pp. 297–330. ISSN: 2573-5144. DOI: `10.1146/ANNUREV-CONTROL-100819-063206`.

[10] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[11] Takayuki Osa et al. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends ® in Robotics* 7 (2 2018), pp. 1–179. DOI: `10.1561/2300000053`.

[12] Jonathan Ho and Stefano Ermon. "Generative adversarial imitation learning". In: *Advances in neural information processing systems* 29 (2016), pp. 4565–4573.

[13] Stéphane Ross et al. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[14] Boyuan Zheng et al. "Imitation Learning: Progress, Taxonomies and Opportunities". In: *arXiv preprint arXiv:2106.12177* (2021).

[15] Yiren Lu and Jonathan Tompson. "ADAIL: Adaptive Adversarial Imitation Learning". In: *CoRR* (2020).

[16] Aude Billard et al. "Survey: Robot Programming by Demonstration". In: Springrer, 2008, pp. 1371–1394. ISBN: 978-3-540-23957-4. DOI: `10.1007/978-3-540-30301-5_60`. URL: `https://infoscience.epfl.ch/record/114050`.

[17] Yuan Guan, Ning Wang, and Chenguang Yang. "Review of the techniques used in motor☐cognitive human☐robot skill transfer". In: *Cognitive Computation and Systems* 3 (3 Sept. 2021), pp. 229–252. ISSN: 2517-7567. DOI: `10.1049/ccs2.12025`.

[18] Aude Billard and Daniel Grollman. "Imitation Learning in Robots". In: *Encyclopedia of the Sciences of Learning* (2012), pp. 1494–1496. DOI: `10.1007/978-1-4419-1428-6_758`. URL: `https://link.springer.com/referenceworkentry/10.1007/978-1-4419-1428-6_758`.

[19] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. "Movement imitation with nonlinear dynamical systems in humanoid robots". In: *Proceedings - IEEE International Conference on Robotics and Automation* 2 (2002), pp. 1398–1403. ISSN: 10504729. DOI: 10.1109/ROBOT.2002.1014739.

[20] Auke Jan Ijspeert et al. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013), pp. 328–373.

[21] Bin Fang et al. "Skill learning for human-robot interaction using wearable device". In: *Tsinghua Science and Technology* 24.6 (2019), pp. 654–662.

[22] Ravi Prakash Joshi, Nishanth Koganti, and Tomohiro Shibata. "A framework for robotic clothing assistance by imitation learning". In: *Advanced Robotics* 33.22 (2019), pp. 1156–1174.

[23] Luka Peternel and Arash Ajoudani. "Robots learning from robots: A proof of concept study for co-manipulation tasks". In: *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE. 2017, pp. 484–490.

[24] Matteo Saveriano et al. "Dynamic Movement Primitives in Robotics: A Tutorial Survey". In: *arXiv preprint arXiv:2102.03861* (2021). DOI: 10.1177/ToBeAssigned. URL: www.sagepub.com/.

[25] Karl Glatz. *Adaptive Learning from Demonstration using Dynamic Movement Primitives*. 2012.

[26] Fares J Abu-Dakka et al. "Adaptation of manipulation skills in physical contact with the environment to reference force profiles". In: *Autonomous Robots* 39.2 (2015), pp. 199–217.

[27] Aleš Ude et al. "Orientation in cartesian space dynamic movement primitives". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 2997–3004.

[28] Fares J Abu-Dakka and Ville Kyrki. "Geometry-aware dynamic movement primitives". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4421–4426.

[29] Bojan Nemec et al. "Transfer of assembly operations to new workpiece poses by adaptation to the desired force profile". In: *2013 16th International Conference on Advanced Robotics (ICAR)*. IEEE. 2013, pp. 1–7.

[30] Alexandros Paraschos et al. "Using probabilistic movement primitives in robotics". In: *Autonomous Robots* 42 (3 Mar. 2018), pp. 529–551. ISSN: 15737527. DOI: 10.1007/s10514-017-9648-7.

[31] Alexandros Paraschos et al. "Probabilistic Movement Primitives". In: *Advances in neural information processing systems* (2013).

[32] Daniel Schäle, Martin F Stoelen, and Erik Kyrkjebø. "Incremental Learning of Probabilistic Movement Primitives (ProMPs) for Human-Robot Cooperation". In: (2021).

[33] Guilherme Maeda et al. "Learning interaction for collaborative tasks with probabilistic movement primitives". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 527–534.

[34] Dorothea Koert et al. "Incremental learning of an open-ended collaborative skill library". In: *International Journal of Humanoid Robotics* 17.01 (2020), p. 2050001.

[35] Congcong Ye, Jixiang Yang, and Han Ding. "Bagging for Gaussian mixture regression in robot learning from demonstration". In: *Journal of Intelligent Manufacturing* (2020), pp. 1–13.

[36] Sylvain Calinon. "A tutorial on task-parameterized movement learning and retrieval". In: *Intelligent Service Robotics* 9 (2016), pp. 1–29. DOI: 10.1007/s11370-015-0187-9. URL: http://www.idiap.ch/software/pbdlib/..

[37] Sylvain Calinon and Aude Billard. "Incremental Learning of Gestures by Imitation in a Humanoid Robot". In: 2007, pp. 255–262. ISBN: 9781595936172.

[38] Jose Hoyos et al. "Incremental learning of skills in a task-parameterized gaussian mixture model". In: *Journal of Intelligent & Robotic Systems* 82.1 (2016), pp. 81–99.

[39] Weiyong Si, Ning Wang, and Chenguang Yang. "A review on manipulation skill acquisition through teleoperation□based learning from demonstration". In: *Cognitive Computation and Systems* 3 (1 Mar. 2021), pp. 1–16. ISSN: 2517-7567. DOI: 10.1049/ccs2.12005.

[40] Thomas Cederborg et al. "Incremental local online Gaussian Mixture Regression for imitation learning of multiple tasks". In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings* (2010), pp. 267–274. DOI: `10.1109/IROS.2010.5652040`.

[41] Miguel Arduengo et al. "Gaussian-process-based robot learning from demonstration". In: *arXiv preprint arXiv:2002.09979* (2020).

[42] Noémie Jaquier, David Ginsbourger, and Sylvain Calinon. "Learning from demonstration with model-based Gaussian process". In: *Conference on Robot Learning*. PMLR. 2020, pp. 247–257.

[43] Chirag Goyal. *Deep understanding of discriminative and generative models.* `https://www.analyticsvidhya.com/blog/2021/07/deep-understanding-of-discriminative-and-generative-models-in-machine-learning/`. [Accessed: 14-1-2021]. July 2021.

[44] Duy Nguyen-Tuong and Jan Peters. "Local gaussian process regression for real-time model-based robot control". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2008, pp. 380–385.

[45] Daniel H Grollman and Odest Chadwicke Jenkins. "Sparse incremental learning for interactive robot control policy estimation". In: *2008 IEEE International Conference on Robotics and Automation*. IEEE. 2008, pp. 3315–3320.

[46] Jie Yang, Yangsheng Xu, and Chiou S. Chen. "Hidden Markov Model Approach to Skill Learning and Its Application to Telerobotics". In: *IEEE Transactions on Robotics and Automation* 10 (5 1994), pp. 621–631. ISSN: 1042296X. DOI: `10.1109/70.326567`.

[47] Aleksandar Vakanski et al. "Trajectory learning for robot programming by demonstration using hidden Markov model and dynamic time warping". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 42.4 (2012), pp. 1039–1052.

[48] Dana Kulić and Yoshihiko Nakamura. "Incremental learning of human behaviors using hierarchical hidden Markov models". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4649–4655.

[49] Leonel Rozo, Pablo Jiménez, and Carme Torras. "A robot learning from demonstration framework to perform force-based manipulation tasks". In: *Intelligent service robotics* 6.1 (2013), pp. 33–51.

[50] Sylvain Calinon et al. "Learning and reproduction of gestures by imitation". In: *IEEE Robotics & Automation Magazine* 17.2 (2010), pp. 44–54.

[51] Sylvain Calinon et al. "Learning collaborative manipulation tasks by demonstration using a haptic interface". In: *2009 International Conference on Advanced Robotics*. IEEE. 2009, pp. 1–6.

[52] Shun-Zheng Yu. "Hidden semi-Markov models". In: *Artificial intelligence* 174.2 (2010), pp. 215–243.

[53] Dana Kulic, Wataru Takano, and Yoshihiko Nakamura. "Incremental on-line hierarchical clustering of whole body motion patterns". In: *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE. 2007, pp. 1016–1021.

[54] Dana Kulic, Wataru Takano, and Yoshihiko Nakamura. "Online segmentation and clustering from continuous observation of whole body motions". In: *IEEE Transactions on Robotics* 25.5 (2009), pp. 1158–1166.

[55] Dizan Vasquez, Thierry Fraichard, and Christian Laugier. "Growing hidden markov models: An incremental tool for learning and predicting human and vehicle motion". In: *The International Journal of Robotics Research* 28.11-12 (2009), pp. 1486–1506.

[56] Stuart Russell. "Learning agents for uncertain environments". In: *Proceedings of the eleventh annual conference on Computational learning theory*. 1998, pp. 101–103.

[57] Shao Zhifei and Er Meng Joo. "A survey of inverse reinforcement learning techniques". In: *International Journal of Intelligent Computing and Cybernetics* (2012).

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

[59] Saurabh Arora and Prashant Doshi. "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *Artificial Intelligence* (2021), p. 103500.

[60] David Silver, J Andrew Bagnell, and Anthony Stentz. "Learning from demonstration for autonomous navigation in complex unstructured terrain". In: *The International Journal of Robotics Research* 29.12 (2010), pp. 1565–1592.

[61] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. "Maximum margin planning". In: *Proceedings of the 23rd international conference on Machine learning*. 2006, pp. 729–736.

[62] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[63] Zheng Wu et al. "Efficient sampling-based maximum entropy inverse reinforcement learning with application to autonomous driving". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 5355–5362.

[64] Saurabh Arora, Prashant Doshi, and Bikramjit Banerjee. "Online inverse reinforcement learning under occlusion". In: *Proceedings of the 18th International Conference on Autonomous Agents and Multi-Agent Systems*. 2019.

[65] Kai Arulkumaran, Marc Peter Deisenroth, et al. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34 (6 Nov. 2017), pp. 26–38. ISSN: 10535888. DOI: 10.1109/MSP.2017.2743240.

[66] Jens Kober, J. Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *International Journal of Robotics Research* 32 (11 Sept. 2013), pp. 1238–1274. ISSN: 02783649. DOI: 10.1177/0278364913495721.

[67] Athanasios S. Polydoros and Lazaros Nalpantidis. "Survey of Model-Based Reinforcement Learning: Applications on Robotics". In: *J Intell Robot Syst* 86 (Jan. 2017), pp. 153–173. DOI: 10.1007/s10846-017-0468-y.

[68] Richard S Sutton and Andrew G Barto. "Reinforcement Learning: An Introduction". In: (1998).

[69] Hongming Zhang et al. "Taxonomy of Reinforcement Learning Algorithms". In: *Deep Reinforcement Learning: Fundamentals, Research and Applications* (Jan. 2020), pp. 125–133. DOI: 10.1007/978-981-15-4095-0_3. URL: https://link.springer.com/chapter/10.1007/978-981-15-4095-0_3.

[70] Philip Odonkor and Kemper Lewis. "Control of shared energy storage assets within building clusters using reinforcement learning". In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 51753. American Society of Mechanical Engineers. 2018, V02AT03A028.

[71] Andrew Lobbezoo, Yanjun Qian, and Hyock-Ju Kwon. "Reinforcement Learning for Pick and Place Operations in Robotics: A Survey". In: *Robotics* 10.3 (2021), p. 105.

[72] Zihan Ding et al. "Introduction to Reinforcement Learning". In: *Deep Reinforcement Learning: Fundamentals, Research and Applications* (Jan. 2020), pp. 47–123. DOI: 10.1007/978-981-15-4095-0_2. URL: https://link.springer.com/chapter/10.1007/978-981-15-4095-0_2.

[73] Marc Peter Deisenroth, Gerhard Gerhard Neumann, and Jan Peters. "A Survey on Policy Search for Robotics". In: *Foundations and Trends R in Robotics* 2 (2 2011), pp. 1–142. DOI: 10.1561/2300000021.

[74] Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[75] Yudha P Pane, Subramanya P Nageshrao, and Robert Babuška. "Actor-critic reinforcement learning for tracking control in robotics". In: *2016 IEEE 55th conference on decision and control (CDC)*. IEEE. 2016, pp. 5819–5826.

[76] Kai-Wei Chang et al. "Learning to search better than your teacher". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 2058–2066.

[77] Wen Sun, J Andrew Bagnell, and Byron Boots. "Truncated horizon policy search: Combining reinforcement learning & imitation learning". In: *arXiv preprint arXiv:1805.11240* (2018).

[78] S. Mohammad Khansari-Zadeh and Aude Billard. "Learning stable nonlinear dynamical systems with Gaussian mixture models". In: *IEEE Transactions on Robotics* 27 (5 Oct. 2011), pp. 943–957. ISSN: 15523098. DOI: `10.1109/TRO.2011.2159412`.

[79] Jordi Bautista-Ballester, Jaume Vergés-Llahí, and Domènec Puig. "Programming by demonstration: A taxonomy of current relevant methods to teach and describe new skills to robots". In: *ROBOT2013: First Iberian Robotics Conference*. Springer. 2014, pp. 287–300.

[80] Peter Pastor et al. "Online movement adaptation based on previous sensor experiences". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2011, pp. 365–371.

[81] Guilherme J Maeda et al. "Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks". In: *Autonomous Robots* 41.3 (2017), pp. 593–612.

[82] Abdeslam Boularias, Jens Kober, and Jan Peters. "Relative entropy inverse reinforcement learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 182–189.

[83] Yangang Ren et al. "Improving generalization of reinforcement learning with minimax distributional soft actor-critic". In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–6.

[84] Beomjoon Kim and Joelle Pineau. "Socially adaptive path planning in human environments using inverse reinforcement learning". In: *International Journal of Social Robotics* 8.1 (2016), pp. 51–66.

[85] Ioannis Havoutis and Sylvain Calinon. "Supervisory teleoperation with online learning and optimal control". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1534–1540.

[86] Luka Peternel et al. "Adaptive control of exoskeleton robots for periodic assistive behaviours based on EMG feedback minimisation". In: *PloS one* 11.2 (2016), e0148942.

[87] Leonel Rozo, Pablo Jiménez, and Carme Torras. "Robot learning from demonstration of force-based tasks with multiple solution trajectories". In: *2011 15th International Conference on Advanced Robotics (ICAR)*. IEEE. 2011, pp. 124–129.

[88] Ioannis Havoutis and Sylvain Calinon. "Learning from demonstration for semi-autonomous teleoperation". In: *Autonomous Robots* 43.3 (2019), pp. 713–726.

[89] Ioannis Havoutis, Ajay Kumar Tanwani, and Sylvain Calinon. "Online incremental learning of manipulation tasks for semi-autonomous teleoperation". In: Institute of Electrical and Electronics Engineers. 2016.

[90] Luka Peternel, Leonel Rozo, et al. "A Method for Derivation of Robot Task-Frame Control Authority from Repeated Sensory Observations". In: *IEEE Robotics and Automation Letters* 2 (2 2017), pp. 719–726. URL: `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7812599`.

[91] Yudha P Pane et al. "Reinforcement learning based compensation methods for robot manipulators". In: *Engineering Applications of Artificial Intelligence* 78 (2019), pp. 236–247.

[92] Carlos Celemin et al. "Reinforcement learning of motor skills using policy search and human corrective advice". In: *The International Journal of Robotics Research* 38.14 (2019), pp. 1560–1580.

[93] Jens Kober and Jan Peters. "Policy search for motor primitives in robotics". In: *Learning Motor Skills*. Springer, 2014, pp. 83–117.