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 alongside humans, 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 are 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 [8]. 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
Imitation Learning	• 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
Reinforcement Learning	• 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

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 Kroemer, Niekum, and Konidaris. 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. As skill transfer does not allow detecting how the skill of a robot has been learned, and how it has 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. This if followwe by a short description about the potential of combining IL and RL methods, in section 2.4. 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 [4, 9]. 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} [10]
- $T = P(s \mid a, s')$ 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 [9]. 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

One frequently used method for learning the skill policies, is IL. In literature, IL is also referred to as Programming by Demonstration (PbD), Learning from (human) Demonstration (LfD), and apprenticeship learning [1]. For the remaining of this paper, the term IL will be used. The choice of IL over other robot learning methods is particularly compelling when ideal behaviour can neither be scripted (as done in traditional robot programming) nor be easily defined as an optimization off a known reward function (as done in RL), but it can be demonstrated. A disadvantage of IL techniques, is that - due to only learning from demonstrations - the performance is limited by the abilities of the teacher.

The process of learning a skill using IL, consist of three steps: observing an action, representing the action and reproducing the action [11]. 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 [11–13]:

- *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, whcih all 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 [7]. The behaviour of the expert (and eventually the learner) can be described as a trajectory $\boldsymbol{\tau} = [(s_1, a_1), \ldots, (s_n, a_n)]$, 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: Behavioural Cloning (BC) and Inverse Reinforcement Learning (IRL). BC obtains the policies using supervised learning [14]. 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 [15]. IRL describes the policy as the solution of an optimization or planning problem [7]. As this method learns a cost function which prioritizes the entire trajectory over others, compounding errors is not an issue [14]. 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.

Table 2.2: Advantages and disadvantage of the two main IL approaches based on multiple papers [14–17].

	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

Policy Representation
As it is the eventual goal of IL 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 [7]. 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.

The representation can either be on symbolic-level (high-level) or trajectory level (low-level) [18]. In the symbolic representation, the predefined motion elements are organized sequentially. Therefore, it allows for an efficient learning of hierarchy, rules and loop, through an interactive process. On the downside, this method depends on a large amount of prior knowledge to predefine important cues and to segment them efficiently. Trajectory-level representation uses a non-linear mapping between the sensory and motor information. It allows for the results to be a generic representation of the movement. The advantage of this is that it allows for the encoding of different types of signal or gestures. This method does, however, not allow for the reproduction of complicated high-level skills. An illustration of both methods is given in fig. 2.1. Besides using one of these policy representations, research has also been successfully conducted where both types were combined. In [19], high-level learning is done to learn a sequence of action for an assembly task. And the low-level learning is added to learn every path as needed for object manipulation.

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 [7]. An overview of the BC approach is given in algorithm 1. The first step is to obtain the experts' demonstrations $\mathcal{D}$. Different methods for doing so were discussed earlier. Next, an appropriate policy representation has to be chosen. 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 as a supervised learning problem [7]. For such problems, the policy is

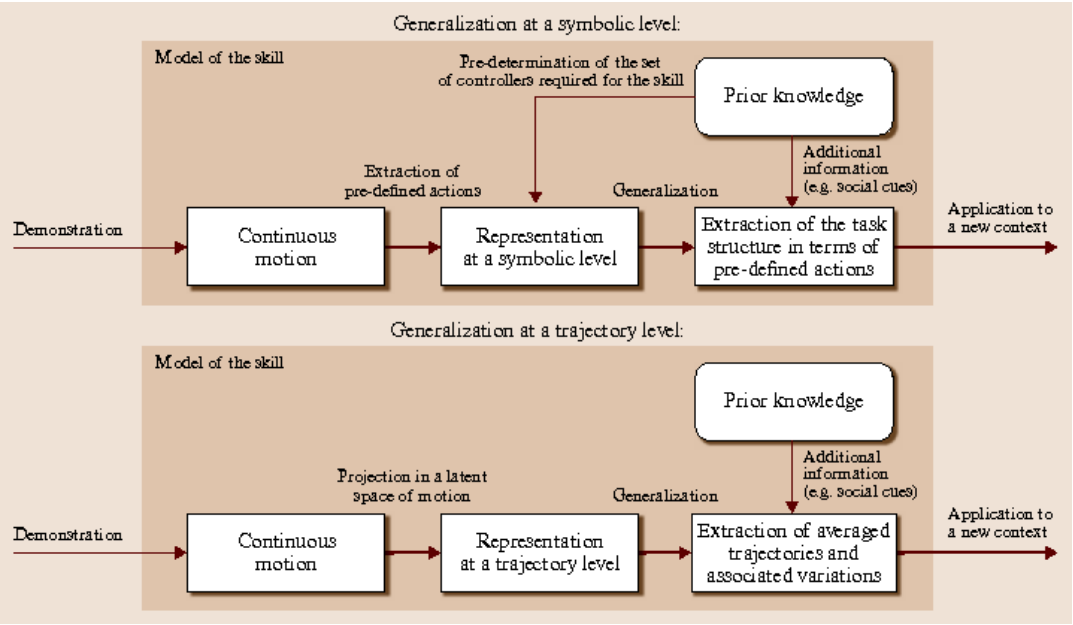


Figure 2.1: Illustration of the different levels of representation for describing the skill [18]. The upper figure shows the model of the skill for a symbolic level, whereas the lower illustrated the trajectory level.

Algorithm 1 Abstract of BC

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

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 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.

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 used method for this representation is Dynamic Movement Primitive (DMP), which was originally introduced by Ijspeert, Nakanishi, and Schaal [20, 21].DMP provides a framework for the motor representation, based on a nonlinear dynamic system [22].

There are two general types of DMPs: discrete and rhythmic [23]. 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 [21]

$$\begin{aligned} \tau\dot{z} &= \alpha_z\left(\beta_z(g-y)-z\right)+f(x), \\ \tau\dot{y} &= z, \end{aligned} \tag{2.2}$$

where y is the state/position of the system, $\dot{y}$ the velocity of the joint trajectory, $\ddot{y}$ the acceleration, z the generalized velocity, 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 [21]. 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, non-linear function. This results into a nonlinear Dynamic System (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:

$$\tau\dot{x} = \alpha_x x, \tag{2.3}$$

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 [23]

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

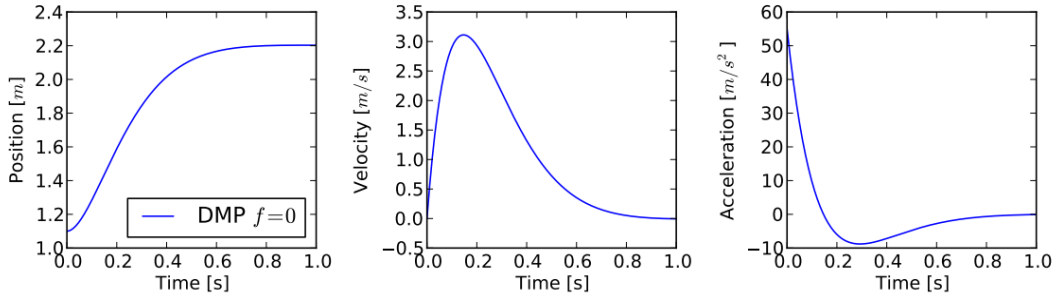


Figure 2.2: Position, velocity and acceleration of an executed DMP without any external force ($f = 0$) [24].

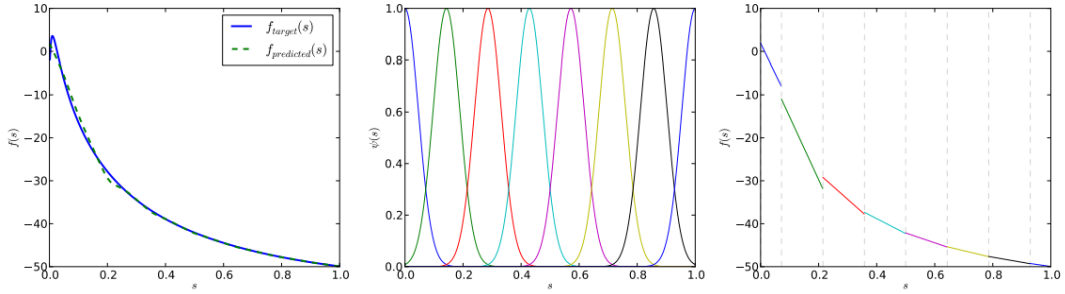


Figure 2.3: Example of the learning of DMP using LWR, where: (left) the target function f_{target} (blue) and the predicted function $f_{predicted}$ (dashed green), (middle) 8 Gaussian functions meaning 8 weights ($|w| = 8$), and (right) the local linear model [24].

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)* [23]. This method is suitable for these types of problems, as it update the existing models fast [25]. 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 [21]. A graphical example of this has been given in fig. 2.3.

Some extension on the original DMP approach exist. The classical DMPs 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].

Typically, DMP is not used for online learning. A reason for this is the usage of only one demonstration. Some papers have extended this original method to allow for online learning. An example is given in [29], where a learning scheme was presented which iteratively updates 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 been introduced by Paraschos 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 τ can be compactly represented by a weight vector w. Here, the trajectory is represented by $\{q_t\}_{t=0\ldots T}$, which is defined by the joint angle q_t over time. Using this will reduce the number of model parameters. The state vector y_t for a single time step, can now be defined

$$y_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \mathbf{\Psi}_t^T w + \epsilon_y, \tag{2.8}$$

where $\mathbf{\Psi}_t = [\psi_t, \dot{\psi}_t]$ 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

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

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 stroke-based or rhythmic [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 ψ_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 Ψ_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

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

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

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

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 contrast to the trajectory distribution generated using DMP (Figure 2.4d), does the one generated using ProMP (Figure 2.4b) 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 the movements for five different targets.

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

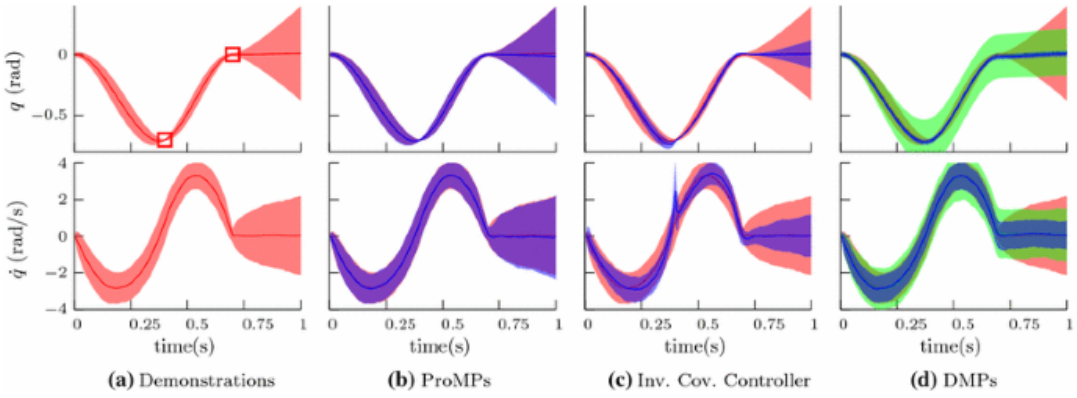
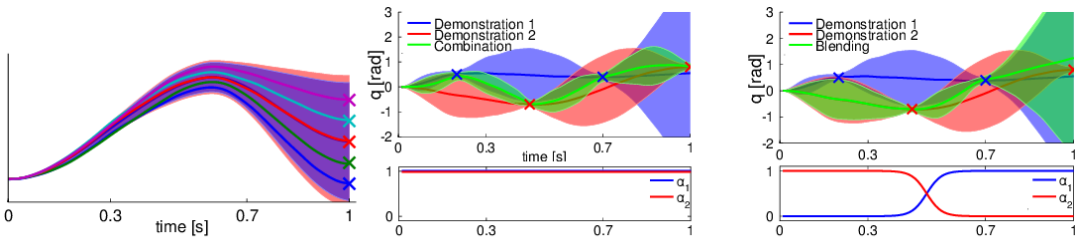


Figure 2.4: The trajectory distribution, showing the joint position (upper plot) and velocity (lower plot). The shaded area is two times the standard variant. The red shaded area shows the demonstrated trajectory [30].



(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.

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 Gaussian Mixture Model (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 k^{th} Gaussian component can be described by

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

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.

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

$$\xi = \begin{bmatrix} \xi_t \\ \xi_s \end{bmatrix}, \quad \mu_k = \begin{bmatrix} \mu_{t,k} \\ \mu_{s,k} \end{bmatrix}, \quad \Sigma_k = \begin{bmatrix} \sum_{t,k} & \sum_{ts,k} \\ \sum_{st,k} & \sum_{s,k} \end{bmatrix}. \tag{2.15}$$

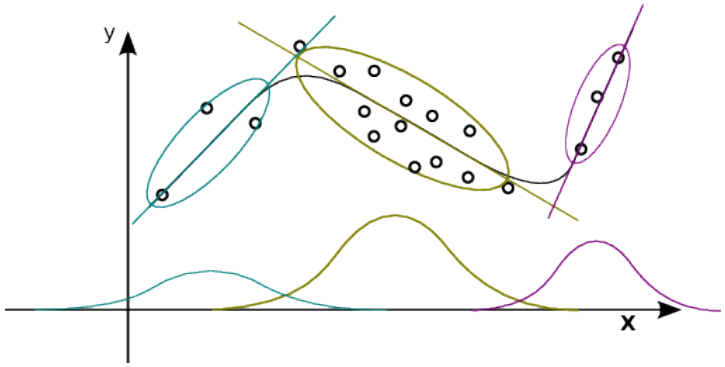


Figure 2.6: Example of trajectory encoding using GMM. The data points are indicated with a dot. The data points are clustered using three Gaussian components. Based on this, the Gaussian distributions are determined (bottom part of the plot), which results in a trajectory which is indicated with a grey, continuous line.

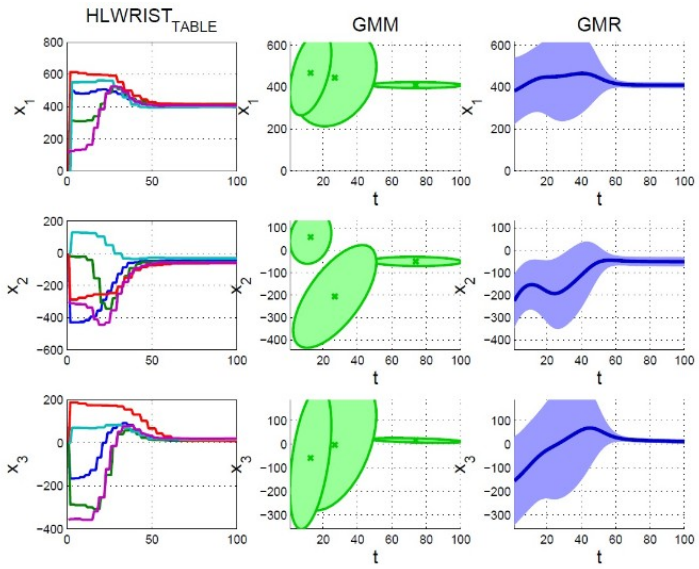


Figure 2.7: Trajectory encoding and generalization using GMM and GMR [35]. It is visible that at the start of execution the constraints are loose as the distribution has a high variance. In contrast, the final part shows that the constraints are quite strict, as the variance of the distribution is low.

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. 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

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

where $k(z_t, z_t')$ is the covariance matrix [7]. 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

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

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

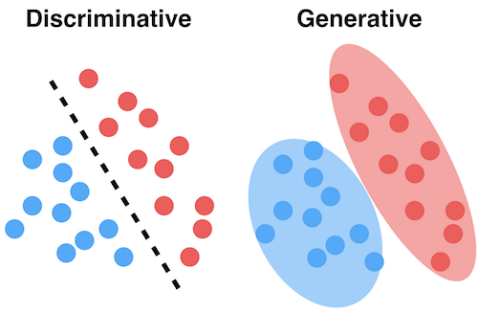


Figure 2.8: Graphical representation of a generative and discriminative model [43].

$$\begin{aligned}\mu\left(\boldsymbol{z}_t^*\right) &= \boldsymbol{K}(z_t^*, Z)^\top (\boldsymbol{K}(Z,Z) + \sigma_n^2 I)^{-1} \boldsymbol{x}_{t+1} = \boldsymbol{K}(z_t^*, Z)^\top \alpha \\ \sigma^2\left(\boldsymbol{z}_t^*\right) &= k\left(\boldsymbol{z}_t^*, \boldsymbol{z}_t^*\right) - \boldsymbol{K}(z_t^*, Z)^\top (\boldsymbol{K}(Z,Z) + \sigma_n^2 I)^{-1} \boldsymbol{K}(z_t^*, Z),\end{aligned} \tag{2.20}$$

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 [7]. 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 $\boldsymbol{O} = \{o_1, \ldots, o_T\}$, the likelihood of observing a given sequence $P(\boldsymbol{O} \mid \lambda)$, can be computed. The observed motion can then be found using

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

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 $[\boldsymbol{O}_1, \ldots, \boldsymbol{O}_T]$, 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

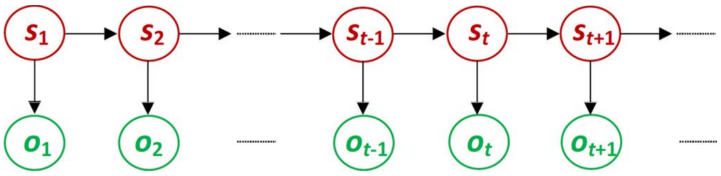


Figure 2.9: Graphical Representation of HMM [47]. With the s_n indicating the states and o_n the observations.

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 [7]. 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 [10]. 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 [9]. 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 [9], 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 [10] 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 [9, 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 [10]. The reward function can then be defined as $R^* = w^{*T}\phi$. In the 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

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

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" [10]. 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 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 [9]. In the remaining of this section, both methods will be discussed.

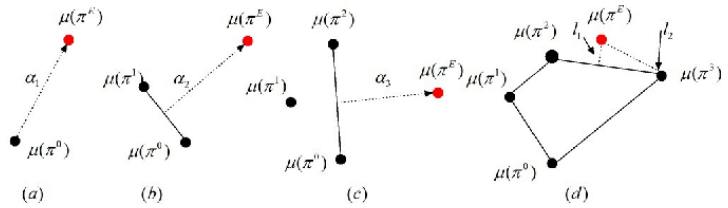


Figure 2.10: Geometric description of policy iteration. The goal is to find a feature expectation $\mu(\tilde{\pi})$, with a different smaller than ϵ compared to the expert's feature expectation μ_E. This difference is indicated as l_1. Each figure *(a)*, *(b)*, etc., is one step further in the iteration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

Max-Margin Inverse Reinforcement Learning
In *maximum margin IRL*, the reward function is identified by maximizing the difference between the best policy and all other policies [12, 58]. The maximization step could be written as the following

$$\min_{t,w} \quad t \tag{2.23a}$$
$$\text{subject to} \quad w^T \mu_E \geq w^T \mu^{(j)} + t, j = 0, \ldots, i-1 \tag{2.23b}$$
$$\|w\|_2 \leq 1. \tag{2.23c}$$

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.

It was found that maximum margin IRL (or MMP), is most frequently used for path planning. Looking at [61], it was already stated that this paper also provided an extension for online learning. In addition, by the set definition, the original method also count as online learning. This, as the model is iteratively being updated, until the margin in reached. Another example is given by Ziebart et al. [62], where MMP is used for autonomous navigation in unstructured terrain.

Besides planning problems, maximum margin IRL has not been found for many other robotic application. A reason for this could be 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 IRL* 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 [7]. 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 (theoretical) drawback of this approach. This has, however, been addressed by the maximum causal entropy IRL [7]. 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 [7]. 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 [7]. 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 [6], also called the policy. This policy will, using a RL approach, be iteratively updated, meaning that typically RL problems take place in an online setting. 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 IL [66]. This makes RL methods able to deal with changing environments [6].

While the principle of RL used for robotic applications is the same as the general approach, applying it comes with some challenges [6]. 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 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

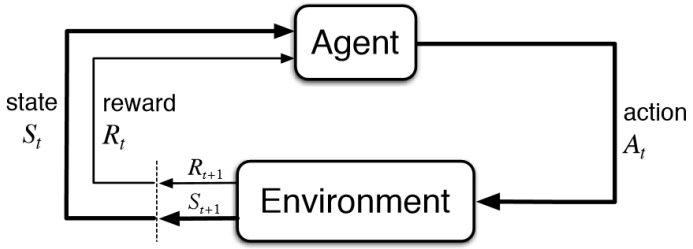


Figure 2.11: The agent-environment interaction in reinforcement learning as defined by Sutton & Barton [67]

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 this method was already given in section 2.2.3. The usage of IRL does have the downside that the advantage of RL compared to, IL of not being limited by the skill of a human expert, does no longer hold.

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 (PS)* approach [66]. 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 optimal policy 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 [66]. 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 [6]. 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 PS. Secondly, it can be optimized using the Lagrange dual formulation, which is known as a value-function based approach. In addition, in 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 [4, 68]. An overview of the taxonomy of RL methods is given in fig. 2.12.

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 [6]. The advantage and disadvantage were described by Polydoros and Nalpantidis [66] and are shown in table 2.3.

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 [6]. 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, 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 [70]. 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

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

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. In the remaining of this section, these classes will be discussed. Before doing so, the idea of bootstrapping should be explained. You are using bootstrapping, is you make an estimation based on an earlier made estimation [65].

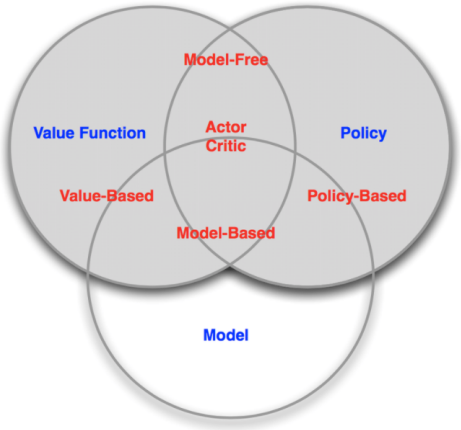


Figure 2.12: Taxonomy of RL methods [69].

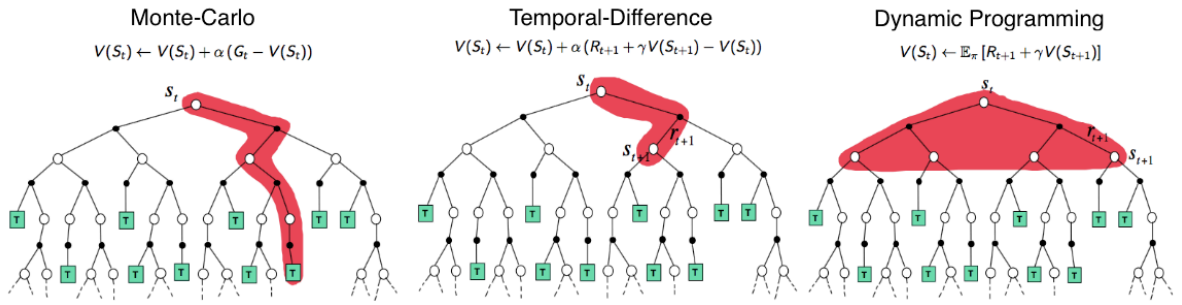


Figure 2.13: Comparison of the backup of three value-function approaches: MC, TD learning, and DP. The hollow circles represent the state value, the closed circles the state-action or action values, and the lines the actions. The tree visualized in the figure represent an example of an entire action-state space. The part which are red indicated represent the part examined by the method. This thus results into two directions: width and depth. The width represents the backup [65], where either a sampling of the actions is done (MC and TD) or expectation of all choices is determined (DP). The depth represents the bootstrapping, which goes from one-step TD to n-step TD learning, where the pure MC approaches do not use bootstrapping at all.

Dynamic Programming based methods

Dynamic Programming (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 [6]. A graphical representation of both method is shown in fig. 2.14.

Policy iteration starts with initializing an arbitrary policy [6]. 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) [66]. 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 [6]. A disadvantage of policy iteration is that the iteration can be a time-consuming task [66].

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 [6, 66].

Monte Carlo

In contrast to the DP approach, is this method model-free [71]. It can be used to replace the policy evaluation step of the value or policy iteration method [6]. 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 [71]. This is done by keeping track and using the frequencies of the transitions and rewards. In the paper of Ding et al. [71] an example of this method is given. MC perform roll-outs by executing

Table 2.3: Advantages and disadvantage of model-free and model-based RL methods [66]

	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

transition function, in other world operating on-policy [6].

Temporal-Difference
This method combines the ideas from DP and MC to estimate the value function [71]. 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 [71]. 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 [6]. To learn, TD uses the error (difference between target value and estimated value) at different time steps [71]. 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 [6, 71], 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.

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 [72]. Therefore, policy search (PS) methods could provide a good alternative. In these methods, the optimal policy is learned directly [66]. 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 [6].

Most PS methods optimize locally around existing policies π, parameterized by a set of policy parameters θ_i [6]. 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. 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. Step-base exploitation uses an exploratory action for each time step, whereas

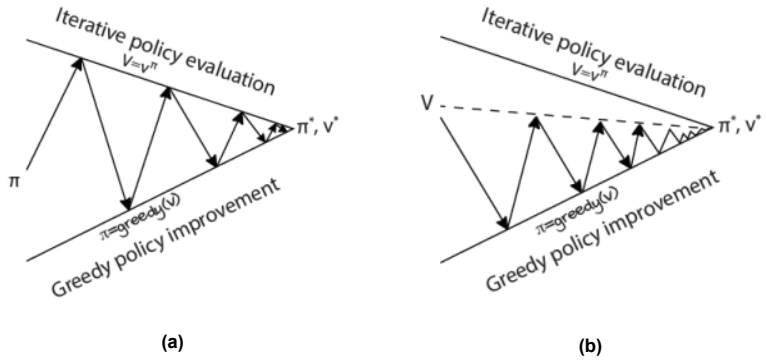


Figure 2.14: Graphical representation of the DP methods. With *(a)* the representation for policy iteration and *(b)* the representation for value iteration.The greedy policy improvement uses the policy which seems to be the best in the short term (after one iteration).

episode-based exploration changes the parameter vector θ only at the start of each episode [72]. The policy update strategies for both model-free and model-based methods are often based on policy gradients (PGs), EM, or information-theoretic insights (Inf.Th.) [6, 66, 72]. In most cases, these methods can both be applied for step-based and episode-based exploration. They will further be explained in the remaining of this section.

Policy gradient
The gradient-based approaches, use a gradient ascent for maximizing the expected return J_θ [6, 72]. 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 [72].

The *finite difference gradient* is the simplest approach and is typically used in an episode-based setting [72]. 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 PG can be computed. Here $p_\theta(\tau)$ is approximated using the sum of the sampled trajectories τ. Algorithms using this approach are `REINFORCE` and `G(PO)MDP` [72].

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 [6]. The idea of this method is to limit the distance between two subsequent distributions, to ensure stability [72]. The main difference with traditional gradient methods, is that they quickly reduce the variance of the policy and, thus, stop exploring. In contrast, natural gradient only gradually decrease the variance, due to their limit in distance, which in the end results in finding the optimal solution faster.

Expectation Maximization
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 Expectation Maximization (EM) algorithm to derive the new policy. EM algorithms do the parameter update by computing it as a weighted maximum likelihood estimate, which has a closed-form solution for most policies [72]. This means that no learning rate is required. 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 [6].

Information-theoretic insights
This last approach tries to update the parameters by ”staying close” to the provided data [72]. This means that the trajectory distribution after the policy update should not jump away from the trajectory distribution before the policy update. Therefore, information-theoretic insights (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 PG algorithm and the Relative Entropy PS.

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 [73] 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.

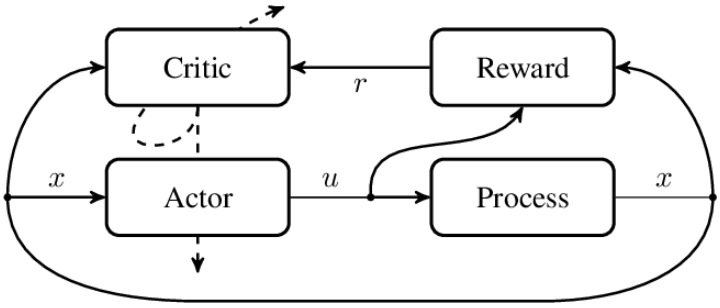


Figure 2.15: Schematic overview of an actor-critic algorithm [73].

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. [73] 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: use an 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 [75]. 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.

Algorithm 2 Actor-critic Algorithm [74]

Initialize λ, γ, α_a, α_c
Initialize ϑ_0, ψ_0
for each trial **do**
    Initialize x_0
    Generate a random initial action u_0
    Initialize eligibility trace $\zeta_0 = 0$
    $t \leftarrow 0$
    repeat
        generate exploration Δu_t
        calculate current action $u_t = \hat{\pi}(x_t, \psi_t) + \Delta u_t$
        apply u_t, measure x_{t+1}
        receive reward $r_{t+1} = \rho(x_t, u_t)$
        $\delta_t = r_{t+1} + \gamma \hat{V}(x_{t+1}, \vartheta_t) - \hat{V}(x_t, \vartheta_t)$ ▷ calculated TD-error
        $\zeta_{t+1} = \lambda\gamma\zeta_t + \frac{\partial \hat{V}(x,\vartheta)}{\partial \vartheta}\Big|_{x=x_t, \vartheta=\vartheta_t}$ ▷ eligibility trace
        $\vartheta_{t+1} = \vartheta_t + \alpha_c \delta_t \zeta_{t+1}$ ▷ update critic parameter
        $\psi_{t+1} = \psi_t + \alpha_a \delta_t \Delta u_t \frac{\partial \hat{\pi}(x,\psi)}{\partial \psi}\Big|_{x=x_t, \psi=\psi_t}$ ▷ update actor parameter
        $t \leftarrow t + 1$
    until t = maximum number of samples T_s
end for

In [76], 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 Sun, Bagnell, and Boots did implement their approach in a simulation, they did not use it in the real-world.

An example where a combined IL and RL approach was used for policy learning, is [77]. For one of their experiments, Celemin et al. found an initial policy using the IL method ProMP. From where, they used, among other things, PS to further learn the policy. In another experiment, they did not compute this initial policy. Comparing the results show a significant improvement while using the method which initially compute the policy using ProMP. This experiment will further be expanded on in section 3.3.2.

While research has shown promising result in combining IL and RL, it is still quite limited. Further research should be done comparing IL and RL to a combined approach.

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 2.4. Here, the different robot teaching methods are compared based on multiple criteria, which will be expanded on in the remaining of this section. It should be noted that the combined IL with RL is not taken into account in this comparison, as not enough information could be found to give a comprehensive results for this method. The 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 fact-based comparison, further research should be conducted.

The *computational efficiency* describes the amount of data and computational time, required to converge to a solution [12]. 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 method, BC often only have to do one iteration per demonstration. There is, however, a

Table 2.4: 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	+/-	+/-	+	+	+/-	+/- [1]	+
RL	Value function	+/-	- [2]	+/-	-	-	-	+
	Policy Search	-	-	+/-	-	-	+	+
	Actor-Critic	-	+/-	+/-	+/-	-	+ [3]	+

[1] *Assumption based on information of IL and RL methods.*
[2] *Assumption based on the information of RL methods.*
[3] *Assumption based on the information of PS method.*

difference in computational time between the different BC method. For example DMP just have to update one parameter [7]. Whereas, GP has to compute a new covariance matrix (which can be large for high-dimensional inputs) and has to invert this [41]. 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 [4]. 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. The reason for this, is 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. For example, DMP is statically asymptotically stable [78], meaning it has a stable attraction to a target position, whereas ProMP [7], 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, one would expect a good stability. However, this is not necessary the case as most method depend on multiple conditions. PG requires the setting of a learning rate, which, if chosen incorrectly, results in unstable behaviour [72]. 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. Another example is the Inf.Th. principle, which states that in order to provide a stable behaviour, the distance between the old and new trajectory distribution, should be bounded. In [70], a solution to increase the stability was presented - replay. During replay, the robot is initiated in states which are not stored, which results in improvement of the exploration efficiency, and the stability. This idea of replay is not necessary for the actor-critic algorithm `A3C` to ensure stability, as it allows for multiple agents to asynchronously explore different policies in parallel [70].

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 [12]. 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]). 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 [4]. BC method often generalize quite well. However, in some cases some extensions are required [7]. An example is ProMP, which learns 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 [70]. 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 [70]. There have been PS algorithm which generalize relatively well. Most of these are episodic-based policy evaluation strategies [72]. Even though some examples can be found in the literature on generalizability of RL, applying this is not frequently done.

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 the system, 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 they are often model-free [7]. 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. Implementing RL methods, is often more difficult than IL. In addition to the difficulty of determining a reward function and a model of the environment, running RL methods can take multiple iterations, which quite costly [6]. A solution for this, is to first run the system in simulation. 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 chosen 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 [4]. The same cannot be said for value function methods.

The last criterium relates to *online* learning. For each method, an online implementation can 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 literature can be found on online implementations of BC methods [25, 29], but as stated they require some adaptations of the original method. IRL uses 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. In contrast to the IL approaches, offline learning using RL not common [70]. Instead, RL is in most cases used in an online setting, where the policy is iteratively updated based on the sensory feedback [6].

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. In addition, these methods are often computational efficient as only a small amount or in some cases just one, has to be done to find a solution. However, due to these demonstrations, the skill of the robot is limited by the skill of the expert. Often, 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.
- **RL methods:** 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.
- **Combining IL and RL:** The idea behind combining IL and RL is to use both of their advantages in one approach: the computational efficiency of IL and the ability to exert a human skill by using RL. While research seems to be promising, the literature combining these approaches still seems to be quite limited.