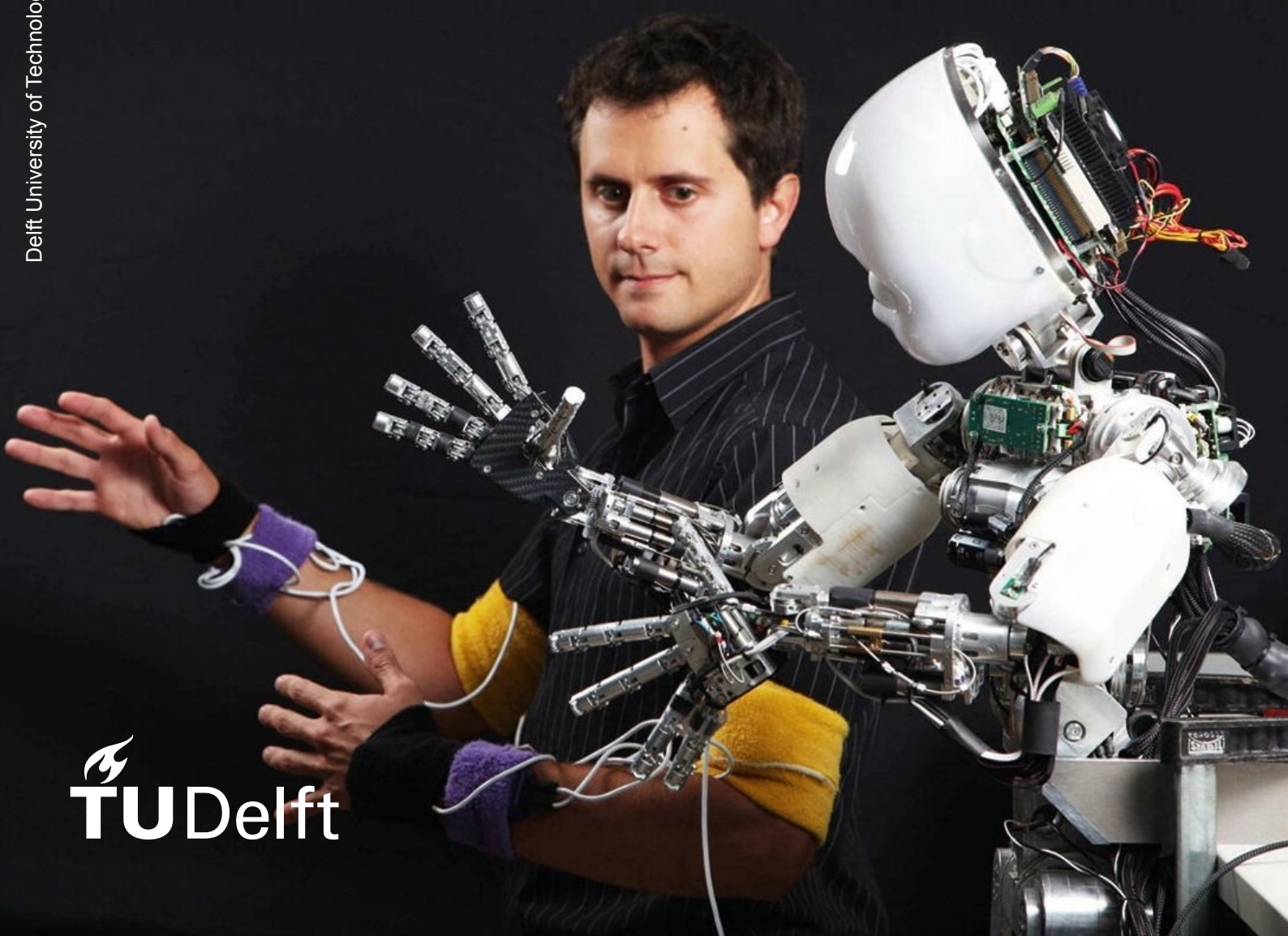
Robotic skill learning

How a novice robot becomes an expert

Literature Study

R.E.S. Maesen



Delft University of Technology

TU Delft

Robotic skill learning

How a novice robot becomes an expert

by

R.E.S. Maesen

Author:	R.E.S. Maessen
Student number:	4564200
Supervisor:	Dr. ir. L. Peternel
	Prof. dr. ir. D.A. Abbink
Institution:	Delft University of Technology
Study:	Msc. Robotics
Date:	Friday 11th March, 2022

Cover Image: Demonstration of the skill with simultaneous reproduction on the robot [1].

TU Delft

Summary

In the past, the robot's behavior was directly programmed which had the disadvantage of not being able to adapt to changes in the environment, as each action had to be predefined. Therefore, the topic of robotic learning has increased in popularity. In this report, the idea of robotic learning will be examined, this is done by trying to answer the following research question:

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

The scope of this research is limited using two aspects. Firstly, only online learning method will be examined, meaning the model representing the behavior of the robot will somehow incrementally change. And secondly, no deep learning approaches will be taken into an account, as the goal is to examine the learned skill and to understand why a skill has changed. Deep learning does not allow for this understanding, as it is a black box system.

To answer the research question, two topics are discussed, each relating to a part of this question. Firstly, in chapter 2 method enabling continuous skill learning will be examined. And secondly, in chapter 3, the changes of the learned skill during the learning process will be discussed.

Two types of skill learning methods were identified: *Imitation Learning (IL)* and *Reinforcement Learning (RL)*. IL learns by imitating an expert based on demonstrations. The two broad IL categories are *Behavioral Cloning (BC)* and *Inverse Reinforcement Learning (IRL)*. The former tries to learn a policy by directly mapping from the state to the trajectories/action, whereas the latter uses the demonstrated date to compute a reward/cost function, which then can be used (using RL) to compute a policy describing the behavior of the robot.

RL learns a policy by interacting with the environment and using a reward function to determine how good a certain behavior is. The two main RL categories are *value function* and *policy search (PS)*. The value function approach tries to find an optimal policy by iteratively optimizing the value function, whereas PS directly learns the policy. Recently, a new type of RL method has been introduced: actor-critic RL, which combines the advantages of both methods by separating the policy (actor) and the value function (critic).

Both IL and RL have their disadvantage. The skill obtained using IL is limited by the skill level of the human expert, and RL is often computationally expensive, as a lot of iterations are required to obtain a solution. Therefore, a possible solution would be to combine these two methods. While research on this topic is still limited, some examples have been found. One obtains an initial policy using IL which is the starting point of the RL method.

In section 2.5, a comparison was made between different IL and RL methods (table 2.4). They were compared using several criteria: computational efficiency, stability, smoothness, generalizability, how difficult they are to implement, ability to deal with high-dimensional input, and whether they were originally designed for online learning.

For continuous learning, multiple learning outcomes were identified, where the one further examined is policy. The continuous learning using policy can be examined based on the used state input (to learn a policy) or the actions taken (due to the learned policy), and the usage of a reward/cost function. To obtain a policy different types of state inputs can be used, e.g. position, joint angle, or force. Position and joint angles work well for static environments, but some problems occur when needing to adapt to the environment (or when they need to generalize). In some cases, additional feedback is used to allow for better adaptation to the environment and generalization of the task. Force is better at dealing with small changes in the environment. An example is a grasping task where the object is displaced only a small fraction, the force feedback allows for noticing this displacement and therefore adapting

to it. However, if the displacement of the object is larger, and the robot does not even touch it, some additional feedback is necessary, e.g. vision.

Research, using action to examine the skill learning of a robot, showed the potential of using a collaborative task for skill learning. Due to not exactly mimicking the behavior of the expert, the robots' behavior can mutate compared to the experts' behavior.

Experiments where a reward/cost function was used to learn, showed the importance of choosing a learning setting for a specific task. Where one method can work well for certain cases, this does not necessarily have to be the same for other situations. Additionally, the usefulness of combining IL and RL was examined, which showed that combining them would increasingly reduce the computational time when compared to only using RL. In addition, the same research has shown the relevance of choosing an initial policy for RL, again resulting in faster convergence.

Based on the literature, it was concluded that both IL and RL allow for continuous skill learning. Here, the IL methods provide a reliable result that is evenly good as that of the expert, whereas RL is less reliable but has the potential of finding better solutions than IL.

Contents

List of Abbreviations

AC actor-critic.

BC Behavioral Cloning.

COACH Corrective Advice Communicated by Humans.

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

EM Expectation Maximization.
EMG electromyography.

GMM Gaussian Mixture Model.
GMR Gaussian Mixture Regression.
GP Gaussian Process.
GPR Gaussian Process Regression.

HMM Hidden Markov Model.
HSMM Hidden Semi-Markov Model.

IL Imitation Learning.
ILWR incremental LWR.
Inf.Th. information-theoretic insights.
IOC Inverse Optimal Control.
IRL Inverse Reinforcement Learning.

LfD Learning from (human) Demonstration.
LGP Local GP.
LWPR Locally Weighted Projection Regression.
LWR Locally Weighted Regression.

MC Monte Carlo.
MDP Markov Decision Process.
MMP maximum margin planning.
MP Motion Primitive.
MPC model predictive control.

PbD Programming by Demonstration.
PG policy gradient.

PiH peg-in-hole.
ProMP Probabilistic Movement Primitive.
PS policy search.

RBF Radial Basis Function.
RL Reinforcement Learning.

SOGP Sparse Online Gaussian Processes.

TD Temporal-Difference.
TP task-parametrized.
TP-GMM TP GMM.
TP-HSMM TP HSMM.

List of Figures

List of Tables

1 Introduction

Traditionally, robotic skills were directly programmed, which meant that in one code was created such that the robot was prepared for all possible scenarios. This code usually consists of hundreds or even thousands of parts, each describing of a certain type of action [2]. While for some shielded environments this can still be useful, robots nowadays often have to execute alongside humans. This demands a high safety level and the ability of robots to adapt to uncertainties in the environment, where the latter is especially relevant, as humans are unpredictable. In addition, it can be desired for robots to execute a similar task in a different environment or maybe even different task while using a set of already known skills, which requires the robot to have some generalizability capabilities, which direct programmed robot do not have. Due to the limitations of the direct programming approach, robotic learning has become a topic of interest in research.

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

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

To narrow the scope of this research, two decisions were made. First, robotic learning can either be on- or offline, where in this report the focus will only be on online learning. Before explaining why, a description of this term should be given, as different literature often uses different descriptions. Online learning is seen in this report as any method which allows for some change in the model (dictating the behavior of the robot), while executing a task. This can both that the robot executes a behavior during the learning stage, where it is seen that the behavior in iteration 1 is different from that in iteration 2. Another possibility is when the model is updated, but this cannot be seen in the real world.

In offline learning, the data is acquired in a demonstration phase, after which the task is stopped for the learning process. During this learning process, a model is trained, which can be used in an autonomous phase to let the robot execute certain tasks. The advantages of this method are that the operator can select the data or adjust it, to achieve the best learning performance [3]. A downside to offline learning is that no direct feedback about the behavior of the robot, resulting from the learned model, is available. This can partly be addressed by providing a variety of data (e.g. different tasks or different starting points) or by correcting the skill during the autonomous stage, but this again requires a high level of robot intelligence to recognize the human device or advise a strategy to correct its performance on the sensorimotor level. Especially for complex tasks, this can be time-consuming and sometimes even unsuccessful. Therefore, online learning can be more appropriate. This method does not make a distinction between the demonstration and learning phases, but instead lets the robotic skill gradually form during demonstration [3]. An advantage of this approach is that the transition between the learning phases, where the robot's learns a certain skill, and the execution stage, where it can autonomously perform a task, can be direct and autonomous. In addition, does this approach allow for useful feedback about the performance while the robot's skill is being constructed. Another

motivation for online model learning is that, during learning, it is often not possible to cover the complete state-space with data beforehand. Instead, only the interesting state-space regions are known during execution [4]. Therefore, online learning will require the incremental acquisition of knowledge and, possibly, even partial forgetting of the recorded information in order to cope with errors as well as change.

Besides offline learning, will deep learning also be excluded from this research. The main reason for this is that it was decided to be of importance to make the learning process of the robot understandable for humans. Deep learning uses a model, which consists of some structured internal scheme (a neural network), to represent the behavior of a robot [5]. While being updated, one cannot completely understand what is going on. Therefore, if a robot executes some unexpected or undesired behavior, the underlying characteristic of its skill cannot be explained as it is a black bock system. In addition, it can be preferable for some real-life tasks to be represented by a simple policy, instead of a complex model [6].

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

2

Skill Learning Methods

An important goal of robotics is to allow robots to perform certain tasks in a smooth and natural way. In the past, this behavior was often programmed using a direct approach, also known as a *"hard-coded"*. While this approach is useful for well-known environments, problems arise when this is not the case. Directly programmed robots are, for example, unable to autonomously adapt to changing environments as all behavior has to be manually programmed. This is especially problematic when these robots work alongside humans, as safety is a top priority. But even when this is not the case, adapting to the environment prevents the robot from harming itself and/or the environment. Another disadvantage of the direct programming approach is that they do not have generalization capabilities, which described the ability of a system to (based on the existing data/knowledge) deal with other situations, e.g. different conditions, starting points, and tasks. Two learning approaches which tackle these problems are Imitation Learning (IL) and Reinforcement Learning (RL), which both have the goal of finding a policy which best describes a skill. IL uses demonstration, exerted by an expert (someone/something who/which can properly execute a task), to train the robot. An example of this, is where a robot is guided by a human to pick up an object (i.e. move arm robot toward the object, close gripper to grasp the object, and move arm to pick up the object). RL takes a trial-and-error approach, where the environment is explored to perform the desired task as good as possible. An example of this approach is where a robot randomly throws darts towards a dart board, where the goal is to hit the middle. If the dart is not in the middle it will receive discounts in the form of how far it is from the middle, this way it knows whether a throw was good, if it has to do something else, or if an earlier throw might be better, and the model (describing the behavior of the robot) can be updated accordingly. An overview of the advantages and disadvantages of these three approaches is shown in table 2.1.

In this chapter, the focus will only be on learning a policy directly. Another approach for robotic learning is by skill transfer, which enables the robot to transfer a skill, learned in one task, to another task [5]. An advantage of this method is that it can increase the efficiency of learning. However, as skill transfer does not allow detecting how the skill of a robot has been learned, and how it has improved through the learning process, it was decided that it would be excluded for further research.

This chapter is dedicated to presenting the theoretical background behind the two different learning approaches which can 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 frequently used methods. Next, a short description about the potential of combining IL and RL methods is given in section 2.4. In section 2.5, a comparison of the different robot teaching approaches, based on multiple criteria, is given. Lastly, a summary of the chapter is presented in section 2.6.

Table 2.1: Comparison of three methods which can be used to define or acquire the skill of a robot [7–9]. The first method, *direct programming*, is completely manually programmed, 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 to program (every action has to be programmed) • Error-prone • No generalization capabilities
Imitation Learning	• No manual program of behavior • Low number of trials needed • Safety guarantees (if expert's behavior is safe the robots behavior is also safe) • High reliability due to expert data	• Correspondence problem (demonstrator's actuation differs from robots) • Robots skill is limited by skill of expert • Cannot deal properly with suboptimal data (if expert's skill is not optimal, the robot's skill will also not be optimal)
Reinforcement Learning	• No manual program of behavior • Robot can learn tasks when humans cannot demonstrate (e.g. hazardous environment) • 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

2.1. Markov Decision Process

The skill learning method, examined in this chapter, require the learning of a policy that best represents the behavior of a robot. Some skill learning methods (mostly Inverse Reinforcement Learning (IRL) and RL) use MDP as underlying decision-making [5]. This method provide a mathematical framework for modeling decision-making in situations which are partially random and partially under the control of a decision maker. It does this by modelling a discrete time process wherein an agent's actions can stochastically influence its environment [10]. This model 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} [11]
- $T = P(s \mid a, s')$ 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 [10]. 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 actions.

2.2. Imitation Learning

Imitation Learning (IL) is a method which can be used to learn a desired skill. In the literature, IL is also referred to as Programming by Demonstration (PbD), Learning from (human) Demonstration (LfD), and apprenticeship learning [2], 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 behavior can neither be scripted (as done in traditional robot programming) nor be easily defined as an optimization of 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, consists of three steps: observing an action, representing the action, and reproducing the action [12]. For the first step, a demonstration of the desired task, performed by an expert, is required, where the expert is an agent who is assumed to know a certain skill. There are different types of demonstration methods that have been used in literature [12–14]:

- *Kinesthetic 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 behavior 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: Behavioral Cloning (BC) and 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, which all try to solve the same problem. Starting, the goal of IL is to learn a policy π_θ which can reproduce the behavior demonstrated by the experts for performing a certain task [9]. The behavior 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, these can also be represented using the feature $\phi_i = (s_i, a_i)$. To learn a policy, it is assumed that an expert is exerting a certain policy π^E. As this policy cannot directly be transferred to the robot, it is instead demonstrated by the expert. The demonstrations are represented by a dataset $\mathcal{D} = (\tau_i)_{i=1}^N$. 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

The different IL approaches are often categorized into two groups: Behavioral Cloning (BC) and Inverse Reinforcement Learning (IRL). An overview of the advantages and disadvantages of both approaches is given in table 2.2. BC obtains the policies using supervised learning [15], by directly mapping from the state input to the action/trajectory [9]. 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 errors [16]. These errors are a result of the difference between the robot's distribution and the expert's distribution, also known as a covariant shift [17].

IRL (also refered to as Inverse Optimal Control (IOC)) learns, instead of a policy, a reward/cost function that prioritizes the entire trajectory over others, making compounding errors not an issue [15]. This method assumes that the expert tries to maximize some reward/cost function during demonstration, and the obtained data is thus used to recreate this function [9]. After learning this function, RL is implemented to obtain a policy which maximizes the expected return.

Policy Representation

As it is the eventual goal of IL to define a policy that best represents the behavior of the expert, an important consideration is how this policy should be represented. The representation should be chosen such that it captures the desired behavior properly, e.g. a linear or a polynomial function [9]. Besides,

Table 2.2: Advantages and disadvantage of the two main IL approaches based on multiple papers [15, 16, 18, 19].

	Advantages	Disadvantages
BC	• Computationally efficient • Good performance in high-dimensional space	• Compounding error caused by covariant shift • Often large amount of data required for stable policy • Bad performance in dynamical and suboptimal space
IRL	• No compounding error • Good performance in suboptimal space	• Computationally expensive (makes impractical for real-world) • Bad performance in high-dimensional space

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 behavior better, however, it will in most cases also lead to an increase 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) [20]. An illustration of both method is given in fig. 2.1. In the symbolic representation, the predefined motion elements are organized sequentially, therefore allowing for efficient learning of hierarchy, rules, and loops 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.

The 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, which has the advantage that the encoding of different types of signals or gestures are allowed. This method does, however, not allow for the reproduction of complicated high-level skills.

In addition to using one of these policy representations, research has also been successfully conducted where both types were combined. In [21], high-level learning is done to learn a sequence of actions for an assembly task and low-level learning is added to learn every path as needed for object manipulation.

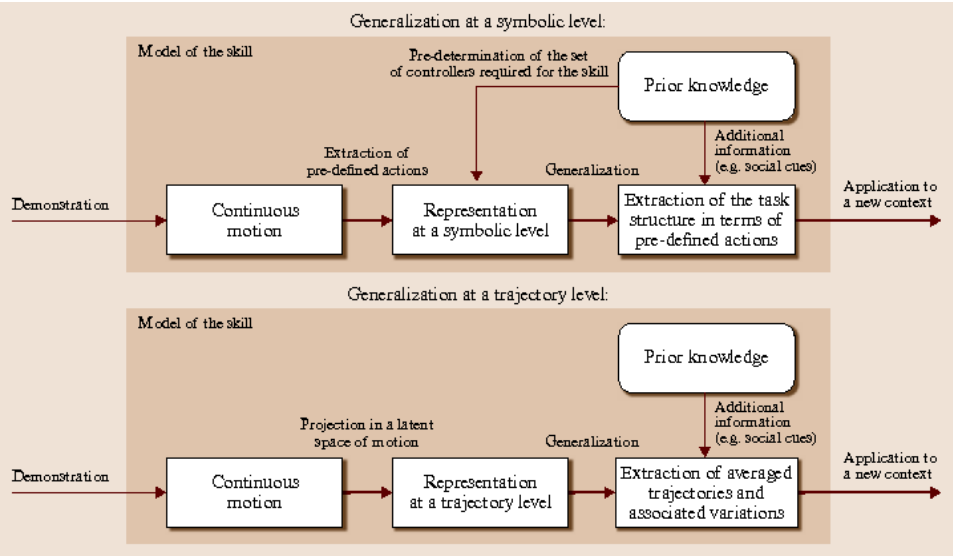


Figure 2.1: Illustration of the different levels of policy representation for describing the skill [20]. The upper figure shows the model of the skill for a symbolic level, whereas the lower illustrated the trajectory level.

2.2.2. Behavioural Cloning

Behavioral Cloning (BC), the simplest form of IL, tries to learn the policy by directly mapping from state to the trajectories/actions [9]. An overview of the BC approach is given in algorithm 1. The first step is to obtain the experts' demonstrations $\mathcal{D}$, using any of the earlier described demonstration methods. Next, an appropriate policy representation has to be chosen (low-level or high-level). In addition, an objective function $\mathcal{L}$, which represents the similarity between the demonstrated behaviors 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. Lastly, using the dataset of demonstration, the policy parameters θ can be learned.

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 $\mathcal{D}$
5: **return** optimized policy parameters θ

BC problems can be formulated as a supervised learning problem [9], which uses a regression problem to obtain the policy. Regression method try to find a relationship in the data, for continuous data this is a function that approximates the data as accurately as possible [22]. As the found trajectory should be physically feasible, a regression method might not always be the best choice as additional constraints are required, e.g. to exclude configuration in which the robot cannot be. It should be noted that in some cases the constraints implicitly satisfy using a regression method, however, in most cases, it is more convenient to use a policy that explicitly satisfies the constraints. To learn these policies, the regression methods are used in a way that the constraints satisfy.

There are different methods that apply the principle of BC for learning the policy. In the next part of this section, some methods 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. Therefore, if relevant, after explaining the main principles of the method, some extensions will be named which allow for online learning.

Dynamic Movement Primitives

Motion Primitives (MPs) are often used to represent and learn basic movements in robotics. To represent this, Dynamic Movement Primitive (DMP) can be applied, which was originally introduced by Ijspeert, Nakanishi, and Schaal [23, 24]. This method provides a framework for the motor representation, based on a nonlinear Dynamic System (DS), which describes the relationship between the forces acting on a robot mechanism and the accelerations they produce [25].

There are two general types of DMPs: discrete and rhythmic [26]. Discrete DMPs are used to encode a point-to-point motion into a stable DS, 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. For a *discrete DMP*, the transformation system can be written as a first-order notation of damper spring model [24]

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

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 the 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 [24], 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, which results in a nonlinear DS which can be difficult to solve.

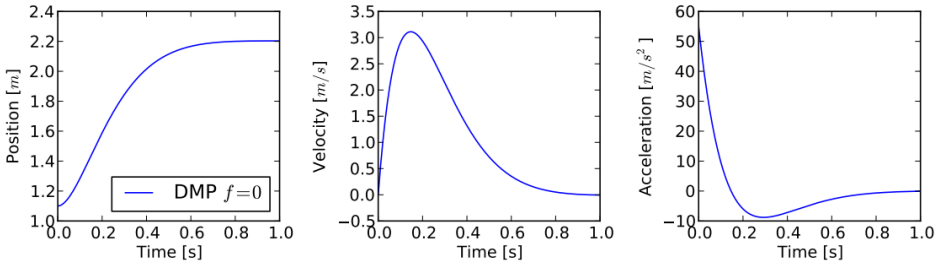


Figure 2.2: Position, velocity and acceleration of an executed DMP without any external force ($f = 0$) [27].

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. To ensure that the robot will follow a smooth trajectory from an initial position y_0 to the final configuration g, $f(x)$ can be defined as a linear combination of N nonlinear Radial Basis Functions (RBFs),

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

where the basis function is defined as $\Psi_i(x) = \exp(-h_i(x-c_i)^2)$, c_i are the centers of the function distribution along the phase of the movement and h_i their widths [26]. For each Degrees of Freedom (DoF), w_i should be adjusted based on the measured data, such that the desired behavior is achieved.

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 dependents on a phase angle 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 is *Locally Weighted Regression (LWR)*, which is also suitable for online learning as it updates the existing models fast[26, 28]. The goal of the regression method, is to find the appropriate weight w_i for the forcing term f. To formulate the function approximation problem, a target function f_{target} should be defined, which is 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 then 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 [24]. A graphical example of this has been given in fig. 2.3.

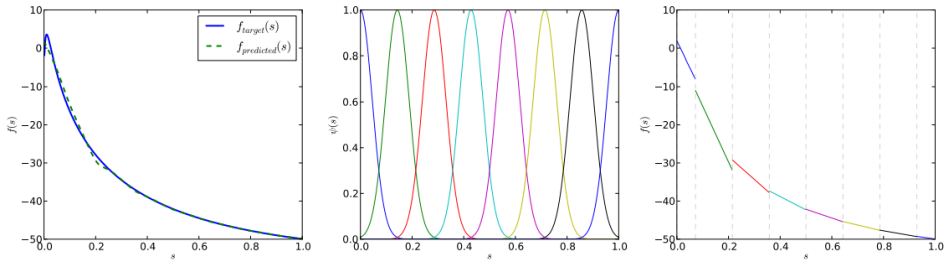


Figure 2.3: Example of the learning of DMP using LWR, where: (left) the target function f_{target} (blue) and the predicted function $f_{predicted}$ (dashed green), (middle) 8 Gaussian functions meaning 8 weights ($|w| = 8$), and (right) the local linear model [27].

Some extension on the original DMP approach exist. The classical DMPs approach is meant for single DoF motions, 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 [29], rotation matrix MDP [30], and symmetric positive definite matrices [31].

Probabilistic Movement Primitives

A disadvantage of DMP is that it can only represent the mean solution [32]. Instead of using such a deterministic approach, a probabilistic approach called *Probabilistic Movement Primitive (ProMP)* has been introduced by Paraschos et al. [33]. This allows for the direct encoding of an optimal behavior for a system with linear dynamics, quadratic costs, and 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 [33].

ProMPs uses multiple trajectories to define a single movement, which automatically results in distribution over the trajectories. To illustrate this method, first a single trajectory τ should be considered. This can, for example, be defined by the joint angle q_t over time: $\tau = \{q_t\}_{t=0...T}$. To compactly represent this trajectory, a weight vector w is used. Now, the state vector y_t (consisting of the joint angle and its derivative) for a single time step is given as a function of this weight vector

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

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 Gaussian noise. The probability of observing the whole trajectory $p(\tau \mid w)$ is then defined as

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

Assuming that different demonstrations of the same movements are slightly different, different weight vectors w_n are needed to represent the n different variants of a movement [34]. 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 [32]. Aside from 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 defines a Hierarchical Bayesian Model, 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 [33]. For stroke-based movements, a Gaussian basis function is used, while for rhythmic movements, a Von-Mises basis function 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 modeled independently. However, often the movement of the multiple joints has to be coordinated. A method for doing this is to introduce the phase variable z_t, which couples the mean of the trajectory distribution [33]. 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 consists 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, the parameters of a single primitive must be easy to acquire from demonstration [33]. To facilitate the estimations of these parameters, a Gaussian distribution $p(w;\theta) = \mathcal{N}(w \mid \mu_w, \Sigma_w)$ over the parameters w is assumed. Again, this could also be done using other types of probability distributions. 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 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 a Hierarchical Bayesian Model 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, whereas fig. 2.5 shows the implementation

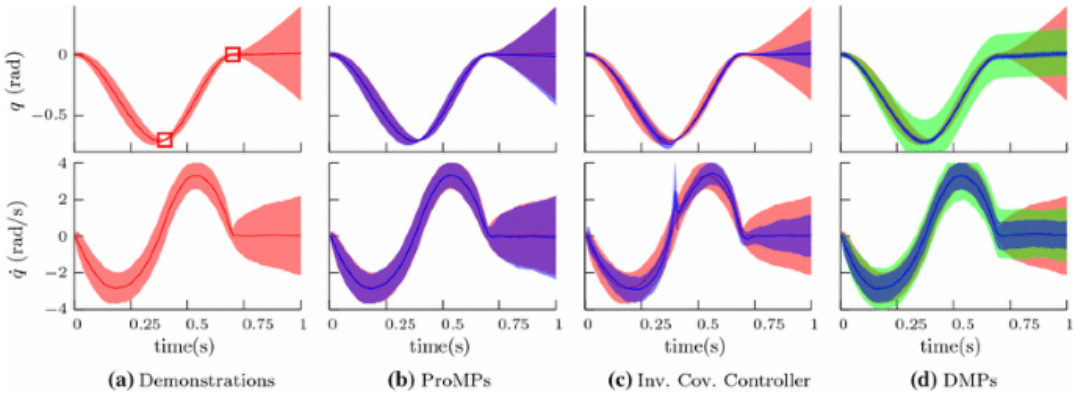
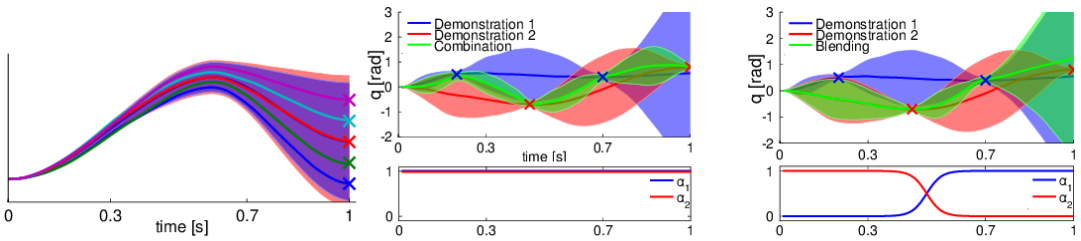


Figure 2.4: The trajectory distribution, showing the joint position (upper plot) and velocity (lower plot). The shaded area is two times the standard variant. The red shaded area shows the demonstrated trajectory [32].



(a) Conditioning (b) Combination (c) Blending

Figure 2.5: The trajectory distribution is encoded using ProMP for multiple goal positions [33]. *(a)* uses conditioning on five different target states, *(b)* combines two ProMPs, and *(c)* blends two ProMPs.

of ProMP for multiple goal positions. The figures show an implementation of conditioning, combining, and blending of ProMPs. Conditioning is used to modulate to new goal states, in the figure it is seen that each new goal state stays within the known distribution. Combing uses a constant activation factor $\alpha \in [0, 1)$) to co-activate each MP in order to result into one distribution. Lastly, blending uses a time-varying activation factor, to also obtain one distribution.

Not much research of online implementation of ProMPs have been found in the literature. A possible explanation for this is that it is a relatively new topic of research. One extension on ProMP that seems to be promising is *Interaction ProMP*, which can be used for collaboration with a human [35]. This extension uses the concept of Interaction Primitives, which is defined as skills that allow robots to engage in collaborative activities with a human partner. By using ProMPs, the Interaction exertion tries to both recognize the action of the human and generates the MP of the robot. This concept was used by Koert et al. in an online learning setting, where the robot had to assist a person in making a salad [36].

Gaussian Process

Gaussian Processes (GPs) is defined as a probability distribution over all functions which can represent a set of points [37]. GP aims to learn a deterministic input-output relationship, up to observation noise, based on a Gaussian prior over potential objective functions [38]. This deterministic model is thus different from the Gaussian Mixture Regression (GMR) approach (later discussed) which results in a generative model. The difference between discriminative and generative models is graphically visualized in see fig. 2.6. While generative models, in most cases, require fewer data, their generalization performance is often poorer than that of discriminative models.

The goal of GP to specify a function $f_i(x_i)$ which can transform the input vector x_i into the target value y_i [40]. In Gaussian Process Regression (GPR) first a GP prior (a prior defining one's belief on a quantity without having looked at the data) is specified by using a mean function $m(x_i)$ and covariance matrix $k(x_i, x_i')$

$$f(x) \sim \mathcal{GP}(m(x_i), k(x_i, x_i')), \tag{2.13}$$

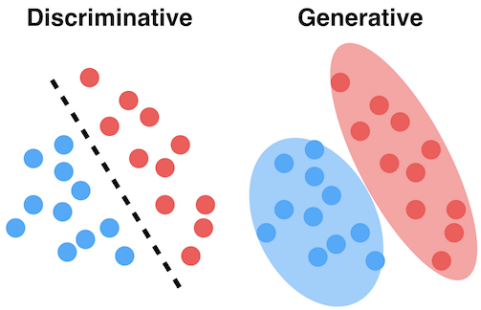


Figure 2.6: Graphical representation of a generative and discriminative model [39].

where x_i is the input signal [9]. Popular approaches for the covariance matrix are the squared exponential covariance function [38] and a Gaussian kernel [40]. To make a prediction $f(x^*)$ for a new input vector x^*, the joint distribution of the observed target values $\boldsymbol{y}$ and prediction value is given by

$$\begin{bmatrix} \boldsymbol{y} \\ f(x^*) \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 I & \boldsymbol{k}(\boldsymbol{X}, x^*) \\ \boldsymbol{k}(x^*, \boldsymbol{X}) & k(x^*, x^*) \end{bmatrix}\right), \tag{2.14}$$

where X is a matrix in which the input vectors x for all training samples are combined. The mean μ and variance σ^2 of the distribution, are then dedicated by

$$\begin{aligned} \mu(\boldsymbol{x}^*) &= \boldsymbol{k}(\boldsymbol{X}, x^*)^\top (\boldsymbol{k}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 I)^{-1} \boldsymbol{y} = \boldsymbol{k}(\boldsymbol{X}, x^*)^\top \alpha, \\ \sigma^2(\boldsymbol{x}^*) &= k(x^*, x^*) - \boldsymbol{k}(\boldsymbol{X}, x^*)^\top \left(\boldsymbol{K}(\boldsymbol{X}, \boldsymbol{X}) + \sigma_n^2 I\right)^{-1} \boldsymbol{k}(\boldsymbol{X}, x^*), \end{aligned} \tag{2.15}$$

where α denotes a prediction vector.

There have been multiple examples found in the literature, which apply the principle of GP in an online setting. Nguyen-Tuong and Peters proposed Local Gaussian Process (LGP), which allowed for online learning [40]. This approach combined the fast computation (compared to standard GPR) of local regression with a more accurate regression method (compared to a locally linear method such as Locally Weighted Projection Regression (LWPR)). In [37] a GP-based learning approach was presented, which allowed for generalization over multiple demonstrations, and encoding of the variability along the different phases of the to be executed task. Jaquier, Ginsbourger, and Calinon [38] proposed a GMR-based GP. This combines 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 [41] the Sparse Online Gaussian Processes (SOGP) is compared to LWPR. SOGP allows for data to be processed while it arrives. An advantage of this method, compared to the original 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.

Gaussian Mixture Model and Gaussian Mixture Regression
A combination of Gaussian Mixture Model (GMM) with GMR is frequently used, as using just a Gaussian Regression method to solve IL problems would in most cases not provide enough complexity. This method is robust to noisy data, has a low computation cost, and can capture the correlation of human motion, making it a powerful tool for robot analysis [42]. This technique uses GMM to encode the behavior 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

$$p(s_{t+1}, \xi_t) = \sum_k p(k) \mathcal{N}(\mu_k, \Sigma_k) \tag{2.16}$$

where $p(k)$ is the prior. The k^{th} Gaussian component can be described by

$$\begin{aligned} p(s_{t+1}, \xi_t \mid k) =& \mathcal{N}\left(\begin{bmatrix} \xi_t \\ x_{t+1} \end{bmatrix} \mid \begin{bmatrix} \mu_{\xi,k} \\ \mu_{x,k} \end{bmatrix}, \begin{bmatrix} \Sigma_{\xi,k} & \Sigma_{\xi x,k} \\ \Sigma_{x\xi,k} & \Sigma_{x,k} \end{bmatrix}\right) \\ =& \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}\left((\xi_t - \mu_k)^T \Sigma_k^{-1} (\xi_t - \mu_k)\right)}, \end{aligned} \tag{2.17}$$

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

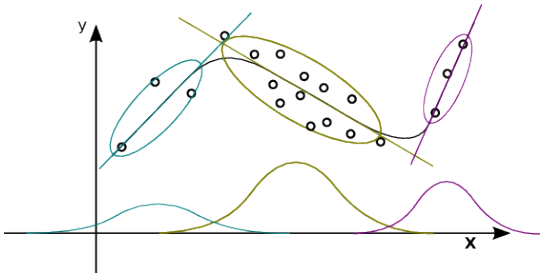


Figure 2.7: Example of trajectory encoding using GMM. The data points are indicated with a dot. The data points are clustered using three Gaussian components. Based on this, the Gaussian distributions are determined (bottom part of the plot), which results in a trajectory that is indicated with a grey, continuous line.

can then be used to reproduce movements [43]. An example of the encoding using GMM is visualized in fig. 2.7.

GMR can be used to reconstruct a general form of the signal [43]. 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 functions 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

$$\xi_= \begin{bmatrix} \xi_t \\ \xi_s \end{bmatrix}, \quad \mu_k = \begin{bmatrix} \mu_{t,k} \\ \mu_{s,k} \end{bmatrix}, \quad \Sigma_k = \begin{bmatrix} \sum_{t,k} & \sum_{ts,k} \\ \sum_{st,k} & \sum_{s,k} \end{bmatrix}. \tag{2.18}$$

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

$$\begin{aligned} \hat{\xi}_{s,k} &= \mu_{s,k} + \Sigma_{st,k} \left(\Sigma_{t,k}\right)^{-1} \left(\xi_t - \mu_{t,k}\right), \\ \hat{\Sigma}_{s,k} &= \Sigma_{s,k} - \Sigma_{st,k} \left(\Sigma_{t,k}\right)^{-1} \Sigma_{ts,k}, \end{aligned} \tag{2.19}$$

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

$$\hat{\xi}_s = \sum_{k=1}^{K} h_k \hat{\xi}_k, \quad \hat{\Sigma}_s = \sum_{k=1}^{K} h_k^2 \hat{\Sigma}_k, \tag{2.20}$$

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. 2.8. 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 the literature, some method were found where GMM and GMR are used in an online setting. Calinon and Billard proposed two method where a model was incrementally updated [44]. The first method, the direct update method, reformulates the problem for a generic observation of multiple data points. 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 [45] proposed three methods for incrementally updating a set of already existing trajectories, which were obtained using task-parametrized Gaussian Mixture Model (TP-GMM) [43]. TP-GMM models local (or relative) trajectories and corresponding local patterns, therefore endowing GMM with

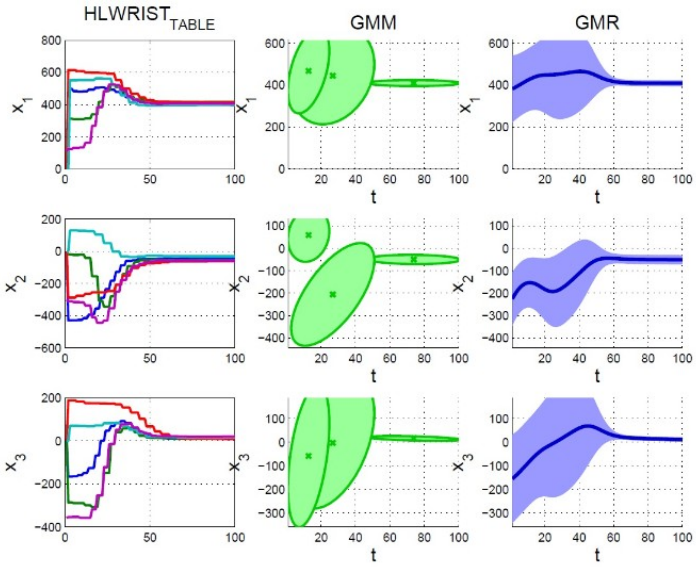


Figure 2.8: Trajectory encoding and generalization using GMM and GMR [42]. It is visible that at the start of execution the constraints are loose as the distribution has a high variance. In contrast, the final part shows that the constraints are quite strict, as the variance of the distribution is low.

better extrapolation performance [46]. 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 proposed method estimates a new model by accumulating the new trajectory and a set of trajectories using the old model. The second method allows for adding parameters for the new trajectories, to the already existing parameters corresponding to the old trajectories. The last method updates the model by the usage of a modified EM algorithm, using the information of the new parameters.

In [47], an incremental local online GMR method was proposed by Cederborg et al. This method allows for the robot to learn a new motor task, by modeling them locally as a DSs. In this paper, sensorimotor context is used to cope with the absence of categorical information both during demonstrations and when reproduction is asked of the system.

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 modeling of a probabilistic transition between discrete states [9]. A reason why HMM is a useful representation of human skill, is because of its ability to discover the nature of the skill [49]. It does this by representing the training data in a statistical 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 [50]. 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

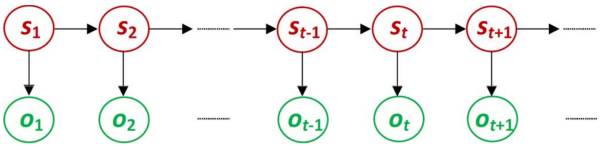


Figure 2.9: Graphical Representation of HMM [50]. With the s_n indicating the states and o_n the observations.

matrices $\lambda = \{A, B, d\}$ [49]. 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 [51].

To allow for incremental learning during observation, the system must be able to deal with new hidden states each time a new MP is learned. In addition, it should also have the ability to learn transition rules incrementally, from partial observations sequences. To do so, an incremental transition rule was proposed by Kulič et al. [51], which is applied each time a partial observation is available.

Using HMM allows for the encoding of the behavior, demonstrated by an expert. There is, however, still a need for a regression method that allows for the reproduction of this behavior. One method for doing so is GMR [52–54]. 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 [9]. HMMs work well when the number of states is relatively low, however, too few states result in the model not being able to reproduce a motion sequence. For robotic application, not being to reproduce a motion sequence can be problematic, 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 [53].

In the literature, some extensions of HMM have been found which allow for online learning. An example is a Hidden Semi-Markov Model, which allows the underlying decision-process to be a semi-Markov chain [55]. A survey paper [55] identified multiple online implementations off Hidden Semi-Markov Model (HSMM), including an adaptive EM algorithm, an online algorithm based on recursive prediction error techniques, and a maximum likelihood estimation algorithm. Another extension is a Hierarchy HMM, which was used in [56, 57], to autonomously segment, cluster, and learn a sequence of full-body MPs from online observations of the full-body human motion. This framework was expanded on in [51], where a higher abstraction HMM was used to learn higher-level ordering between MPs Lastly, a Growing HMM [58] 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 only focus on implementing this concept for trajectory estimation for vehicles.

2.2.3. Inverse Reinforcement Learning

Another form of IL is Inverse Reinforcement Learning (IRL) in which the learner tries to recover a reward function of the environment, based on the expert's demonstrations. The goal is to find a reward that leads the learner to act similarly to the expert, while generalizing well to situations where the expert data is not available [10]. After learning the reward function, RL is used to find a policy describing the behavior of the robot [11]. IRL methods assume that the expert tries to maximize a reward function during demonstrations, making it the aim of the IRL algorithm to estimate this function. To summarize, IRL consist of two steps: 1) reward function estimation and 2) policy optimization based on this reward function. In addition to finding this reward function, implementations have also been found where it is the goal to find a cost function. Another variation, found in literature [10], is to learn from failures instead of success, this approach is referred to as IRL from Failure.

The IRL problem was originally defined by Russel [59]. 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 to that agent;
3) a model of the physical environment (including the agent's body).
Determine: the reward function that the agent is optimizing.

Ng, Russell, et al. [11] defined three main types of IRL algorithms, which are still the most commonly used approaches [60]; 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 are available rather than the desired policy, and will therefore only be considered. By obtaining data from an expert interacting with a MDP (discussed in section 2.1), this approach tries to find a reward function R^*, of which it was assumed that the expert was trying to maximize [10, 61]. This is the opposite of 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 $\mathcal{M}/R$, which is described by a tuple of the form $\langle S, A, T, \gamma \rangle$ [61].

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 for the reward function R can be challenging, most approaches approximately it as a linear combination of the feature [11], which defines is as $R^* = w^{*T}\phi$. In the literature, examples 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. The estimation of the policy $\tilde{\pi}$ should be found, such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$, with ϵ the maximum allowed error. When finding such a policy $\tilde{\pi}$ ($w \in \mathbb{R}^k$), with $\|w\|_1 \leq 1$, 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 $i = 1$.

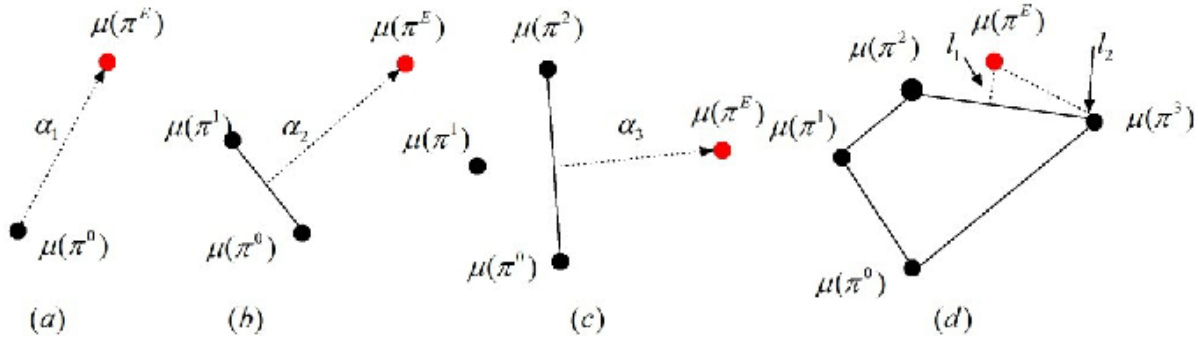


Figure 2.10: Geometric description of policy iteration [60]. The goal is to find a feature expectation $\mu(\tilde{\pi})$, with a difference smaller than ϵ compared to the expert's feature expectation μ_E. This difference is indicated as l_1. Each figure *(a)*, *(b)*, etc., is one step further in the iteration process.

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 of IRL problems is the fact that they are often *ill-posed*, which means that multiple rewards could explain the observations [11, 62]. 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 and therefore making IRL suffer from an ambiguous solution. The *maximum margin* method tries to tackle this problem by converging on a solution that maximizes some margin, but this has the downside that it introduces a bias into the learned reward function, which results in exclusions of meaningful solutions [61]. 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 [10]. In the remainder of this section, both methods will be discussed.

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 [13, 61], with the maximization written as 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 [61].

The apprenticeship learning problem as presented in [61] and eq. (2.23) had as a downside that there was no mechanism for explicitly matching to the experts' behavior and the solution was a stochastic mixture of multiple policies [63]. To address these problems, Ratliff, Bagnell, and Zinkevich proposed maximum margin planning (MMP) [64], which produces a single deterministic solution while also ensuring an upper bound on the mismatch between the demonstrated and planned behavior. Their research also provides an extension for online learning.

In the literature, it was found that maximum margin IRL (or MMP) was mostly used for path planning. Looking at [64], it was already stated that this paper also provided an extension for online learning. In addition, by the set definition, the original method also counts as online learning. This, as the model is iteratively being updated, until the margin in reached. Another example is given by Ziebart et al. [65], where MMP is used for autonomous navigation in unstructured terrain.

Besides planning problems, maximum margin IRL has not been found for much other robotic applications. A reason for this could be the existence of another IRL approach, namely maximum entropy IRL [65]. The maximum margin approach [61] 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 best represents the demonstrated behavior [65]. This allows for obtaining a distribution over behavior, which are parameterized by the reward function [9], now a probabilistic approach can be used to solve the IRL problem [60]. The probability of an observed experts' trajectory τ ($p(\tau \mid w)$

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 $p(\tau \mid w)$ is also affected by the transition probability

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

Now, this approach tries to optimize an estimate of the reward function $\tilde{R}(\tau)$

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

Visible is that there is 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 [9]. 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 [9]. 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 [9], 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 [66] 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 proposed a framework online IRL called incremental IRL, where maximum entropy was adapted such that it was generalized under occlusion [65, 67]. 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 with the environment. 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, an agent can learn to change its behavior based on the rewards it has received [68]. Using this trial-and-error approach, the agent can autonomously discover an optimal behavior (based on a defined metric) for a certain task, also called the policy [8]. As the policy will be iteratively updated, RL typically take place in an online setting. A difference from IL is that, due to the learning of its environment, there is also no need for an external teacher [69], making the learned skill not limited by the skill of the demonstrator.

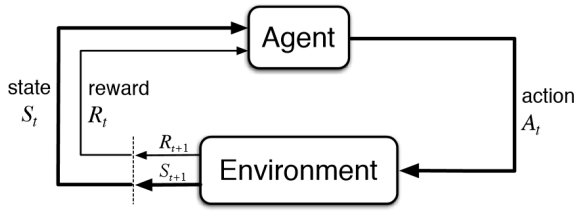


Figure 2.11: The agent-environment interaction of RL problems, as defined by Sutton and Barto [70]

The principle of RL used for robotic application is not different from other types of problems, however, applying it for such a setting comes with some challenges [8]. Firstly, the system has to deal with large amount of data which makes it computationally expensive, this challenge is referred to as the curse of dimensionality. The large dataset is partly due to learning in a high-dimensional space (10 to 30-dimensional space), resulting from the many DoFs of the robots. In addition, it is due to the state and actions being are continuous. The second challenge relates to the fact that the robot is trained in the real-world, which means that it has to deal with real-world challenges like the cost of robots (both robot itself and maintenance), dealing with external factors (e.g. temperature, light conditions, etc.), and the inability to measure everything. This last points relates to the fact that it is often unreasonable to assume that states are completely observable and noise-free, making the problem partially observable. These problems can partially be dealt with by using simulation to train the model. The third challenges is that the algorithm needs to be robust to models that do not capture all the details of the real system, also referred to as under-modeling, and to model uncertainty. If, for example, the real-world application differs from the simulations, the system can diverge and result into unexpected behavior. Lastly, RL algorithms require a well-defined reward system, which 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. As stated earlier, the usage of IRL does have the downside that the skill is being limited by the skill of the expert, which is a drawback that was solved using RL in the first place.

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 by using a trial-and-error approach. This allows for the agent to learn tasks that 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. There are different expression for the expected return, but the main type are either discounted or averaged, and are calculated on a finite or infinite horizon [69]. The discount factor, used in discounted return, determines how much the future is taken into account. Choosing this factor is critical. Policies resulting from optimizing with a small discount factor are myopic (it does not think far a head) and greedy (always tries to immediately find the highest reward, thus not thinking about the long term), which can result in poor performance when caring about the long-term reward [8]. Choosing a low value can also result in unstable behavior, which makes it often not suitable for robotic application. When the discount factor approaches 1, the expected return is set to be averaged. Average rewards have the problem that it cannot distinguish between policies that initially gain a large reward, and those who do in the further. However, this disadvantage is far less critical than one of the discount factors, thus making the average reward setting often more desirable for robotic applications. The average reward return is 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, s and a the state and action, and $R(s,a)$ the reward function depending both on the state and the action. In addition to the named reward function, there are other variants which are also commonly used, namely a reward function which only depends on the state $R(s)$, and a reward function which depends on the state, action, and the state transition s' - $R(s',s,a)$ [8]. 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 [8]. The finding of a policy π can be described as an optimization problem for $J(\pi)$

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

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

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

where $\mathcal{S}$ and A are the state- and action-space, $\pi(s,a) = P(a \mid s)$ the policy written as a conditional probability distribution, and $T(s,a,s') = P(s' \mid s,a)$ is the transitions between state s caused by actions a. Equation (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 general methods were defined. Firstly, the optimal solution can be searched directly in its original, primal problem, which is known as policy search (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. This approach aims to combine the advantages of the value function and PS, by separating the policy (actor) and value function (critic).

Besides the types of algorithms used, there are two more criteria on how to categorize RL problems: model-based or model-free, and on-policy or off-policy [5, 71].

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, whereas the model-free methods not depend on such a model [8]. The advantage and disadvantage were described by Polydoros and Nalpantidis [69] 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 [8]. 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, which is not necessary the case for the off-policy approach. An example of off-policy is where the experience of other agents is used to improve the policy.

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

Table 2.3: Advantages and disadvantage of model-free and model-based RL methods [69]

	Advantages	Disadvantages
Model-based	• Small number of interactions between robot & environment • Faster convergence to optimal solution	• Depends 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

2.3.2. Value-function approaches

The Value Function 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 and a potential, which depends on the future reward achieved using the agent's current policy [72]. 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*. Instead of the value function $V^{\pi}(s)$, the state-action value $Q^{\pi}(s,a)$ can be used. 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 approaches: 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. 2.12. In the remainder of this section, these classes will be discussed.

Dynamic Programming

Dynamic Programming (DP) based approaches, 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 types of DP methods are policy iteration and value iteration [8]. The main difference between these approaches, is that policy iteration starts with initializing a policy, and value iteration with initializing a value function. A graphical representation of both methods is shown in fig. 2.13.

Policy iteration starts with initializing an arbitrary policy [8], after which 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) [69]. 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 no longer change [8].

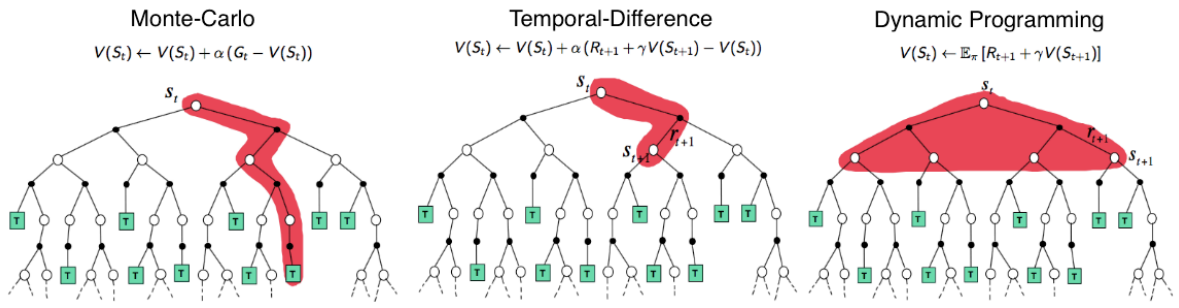


Figure 2.12: Comparison of the backup of three value-function approaches: MC, TD learning, and DP [73]. The hollow circles represent the state value, the closed circles the state-action or action values, and the lines the actions. The tree visualized in the figure represents an example of an entire action-state space. The red part represents the part examined by the method. This thus results in two directions: width and depth. The width represents the backup [68], 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 (making of an estimation based on earlier estimations), which goes from one-step TD to n-step TD learning, where the pure MC approaches do not use bootstrapping at all.

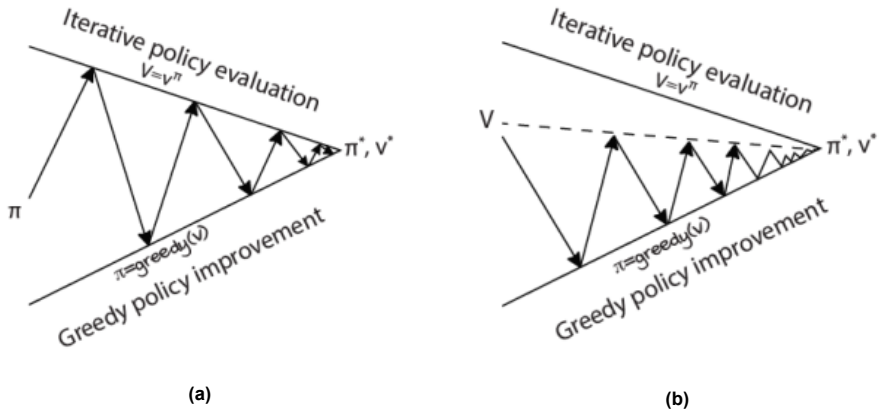


Figure 2.13: Graphical representation of the DP methods [74]. With *(a)* the representation for policy iteration and *(b)* the representation for value iteration.The greedy policy improvement uses the policy which seems to be the best in the short term (after one iteration).

The *value-function* approach starts by initializing a value-function. 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. Instead, this approach updates every time a state is updated [8, 69].

Monte Carlo
Monte Carlo (MC) can be used to replace the policy evaluation step of the value or policy iteration method [8]. In contrast to the DP approach, is this method model-free [75]. The value function V^{π} can be estimated by randomly sampling many trajectories starting from state s, according to the given state transition matrix P [75]. This is done by keeping track and using the frequencies of the transitions and rewards. MC perform roll-outs (single trail run) by executing transition function, in other world operating on-policy [8].

Temporal-Difference
Temporal-Difference (TD) combines the bootstrapping concept of DP with the sample approach of MC to estimate the value function [75]. Bootstrapping (the estimation is based on an earlier made estimation [68]) is used to form a target from the observed return and an estimated state value for the next state [75]. The sample approach is used to estimate the value function. To be precise, sample transition in the MDP are used [8]. To learn, TD uses the error (difference between target value and estimated value) at different time steps [75]. A basic TD method for the updating can be defined as

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

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 [8, 75], 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 algorithms are `SARSA`, which is on-policy, and `Q-learning`, which is off-policy.

2.3.3. Policy Search

A problem with the value function approximation is that it results 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 results in the propagation of error in the value function through to the policy [76]. Therefore, policy search (PS) methods could provide a good alternative. In these methods, the optimal policy is learned directly [69]. Generally, they exist of a set of parameters, which has to be optimized such that the cumulative reward is optimized. Where value function searches 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 policies, which, in other words, means that it is less occupational expensive. PS has been seen as a more difficult approach, in 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 value-function method [8].

Most PS methods optimize locally around existing policies π, parameterized by a set of policy parameters θ_i [8]. 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 strategies, should be made. Step-base exploitation uses an exploratory action for each time step, whereas episode-based exploration changes the parameter vector θ only at the start of each episode [76]. 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.) [8, 69, 76]. 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 [8, 76]. 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, three of them are finite difference gradient, the likelihood ratio method, and natural gradient [76].

Finite difference gradient is the simplest approach and is typically used in an episode-based setting [76]. 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 is 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` [76].

As convergence of the finite difference gradient and likelihood ratio method can be slow, the *natural gradient policy* approaches, which allow for fast convergence, might be advantageous for robotic application [8]. The idea of this method is to limit the distance between two subsequent distributions, to ensure stability [76]. The main difference with traditional gradient methods is that they quickly reduce the variance of the policy and, thus, stop exploring. In contrast, natural gradients 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 requires 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 PS as an inference problem with latent 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 [76]. 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 [8].

Information-theoretic insights
Information-theoretic insights (Inf.Th.) tries to update the parameters by "staying close" to the provided data [76]. This means that the trajectory distribution after the policy update should not jump away from the trajectory distribution before the policy update. Therefore, Inf.Th. approaches bound themselves by the old trajectory distribution $q(\tau)$ and the newly estimated trajectory distribution $p(\tau)$, at each update step. This limits the information loss of the updates, and thus avoids the new distribution convergence to a local optimum. Some methods 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 forms 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. 2.14. Both of these entities are represented as a parametrized function of the state x, with the parameters ψ and ϑ belonging to the actor and critic, 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 [77] use TD for this. The TD-error δ, calculated based on the new state and reward, is used to estimate the actual value function and criticize the actor, by updating the corresponding parameters. If $\delta > 0$, the action is favorable, 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.

Algorithm 2 shows the algorithm of the actor-critic method. While most points can be connected to the image in fig. 2.14, some additional explanation is required to understand this algorithm. First, some coefficients are initialized: 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. Next, the initial actor and critic parameters (ϑ_0 and ψ_0) are initialized. 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 $\delta_{t+1} = \delta_t + \alpha_{c,t} \delta_t \phi(x_t)$, which 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. [77] defined two criteria for the taxonomy of actor-critic methods. 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.

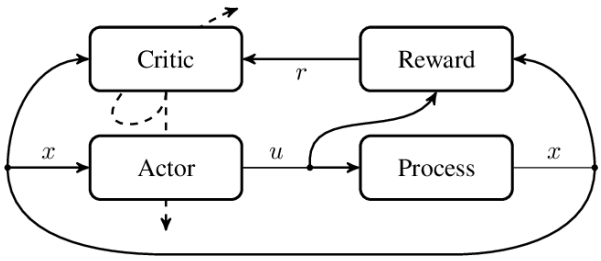


Figure 2.14: Schematic overview of an actor-critic algorithm [77].

Algorithm 2 Actor-critic Algorithm [78]

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

2.4. Combining Imitation Learning and Reinforcement Learning

IL and RL both provide features that make 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 longer. A combination of these two methods could 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 [79]. For doing so, the IRL framework `L2S` was used in combination with a policy-iteration based RL method. As a result, the learned policy would either perform as well as the expert policy (due to IL method) or reach a local optima (due to RL method). While it is possible that the local optima result in a better performance than the experts' policy, it is difficult to quantify this.

In [80], the idea of combining IL and RL was implemented via the idea of reward shaping. For the IL part, access is assumed to be a cost-to-go oracle, which provides an estimate of the expert's cost-to-go during training. The principle of cost-to-go (or reward-to-go) is that a cost is only received after a certain action has been taken. After shaping the cost, the planning horizon of the new MDP would be truncated, followed by a search for a policy that optimizes over the truncated planning horizon. While this approach was implemented 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 [47]. In one of the 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 shows a significant improvement while using the method which initially computes the policy using ProMP. This experiment will further be expanded on in section 3.3.2.

While research has shown promising results in combining IL and RL, it is still quite limited. Further research should be done comparing IL and RL to the 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, where the different robot learning 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 approach is not taken into account in this comparison, as not enough information could be found to give a comprehensive review on this method. The comparison is a summary of information found in literature and assumptions made based on this. The used literature will be referenced in the remaining of this section. 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 [13]. Especially for real-time applications, 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 static, safe environment, however, it could be a problem when adaptation to a (changing) environment is required. IL method often have a higher computational efficiency than RL, due to the fact that only a few (sometimes only even one) iterations are needed to obtain the policy. RL on the other hand, use a trial-and-error approach, resulting in the need of much more iteration. Looking at the different IL method, BC often only have to do one iteration per demonstration, making it in most cases more efficient than IRL who typically have to update their cost function multiple times. A difference in computational efficiency between the different BC methods can be noted. For example, DMP just has to update one parameter [9] whereas GP has to compute a new covariance matrix (can be large for high-dimensional inputs) which also has to be inverted [37]. Looking at the RL methods, the value function methods are known to have a high sample efficiency in comparison to the PS method [5]. This could be translated into the former having a higher computational efficiency than the latter. As actor-critic methods use the bootstrapped value function to reduce the variance of the gradient estimate, they have a similar sample efficiency as the value function.

Stability refers to the ability of the system to deal with perturbations. In cases where perturbation would result in a big difference in the behavior, the system is said to be unstable. Instead, it is desired that the behavior only changes a small fraction or maybe not even at all. The reason why stability is desired, 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 [81], meaning it has a stable attraction to a target position, whereas ProMP [9],

Table 2.4: Comparison of robot teaching methods. It should be noted that this comparison is based on an interpretation of literature. To have a more conclusive analysis, research is required. The methods are compared based on some criteria, here + indicates a high score, +/- an average score, and - a bad score. These scores were given with respect to each other. From left to right, the different criteria are; *Computational efficiency* describing the amount 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 the situation not occurred during the learning process (e.g. new starting point or goal). *Implementation* describes the difficulty of implementing a certain system. *High-dimensional input* states whether a system can deal with a large amount of data. Lastly, *online* describes whether the system was originally implemented for online usage.

	Method	Comp. efficiency	Stability	Smoothness	Generalizability	Implementation	High-dim. input	Online
IL	BC	+	+/-	+	+	+	+/-	-[1]
	IRL	+/-	+/-	+	+	+/-	+/- [2]	+
RL	Value function	+/-	- [3]	+/-	-	-	-	+
	Policy Search	-	-	+/-	-	-	+	+
	Actor-Critic	-	+/-	+/-	+/-	-	+ [4]	+

[1] *Some BC method have been implemented in an online setting, but most of the original methods were ment for offline learning.*
[2] *Assumption based on information of IL and RL methods.*
[3] *Assumption based on the information of RL methods.*
[4] *Assumption based on the information of PS method.*

GMR and GPR [82] do not provide this guarantee. As RL methods generally search a large part of the solution space to find a policy, one would expect good stability, however, this is not necessarily the case, as most methods depend on multiple conditions. PG requires the setting of a learning rate, which, if chosen incorrectly, results in unstable behavior [76]. 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 behavior, the distance between the old and new trajectory distribution, should be bounded. In [72], 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 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 [72].

With *smoothness*, the smoothness of the encoded behavior (often trajectory) is meant. To minimize the risk of damage to the robot, it is desired that the behavior is smooth, without sharp changes [13]. 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 figures in the literature, where for each approach multiple papers were used (for BC [33, 37, 44, 53, 83, 84] and for IRL [64, 85]). As IL methods try to imitate the expert, which is assumed to have a smooth behavior, the resulting behavior 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 stability, the usage of a bad trajectory would result in rough behavior.

Generalizability is the ability of a system to perform well when using different settings/environments than used during the training of the system [62]. Examples of this are states and action unobserved in the demonstration, starting the task at a different initial state, and reusing skills in different problem settings [5]. BC method often generalize quite well. However, in some cases, some extensions are required [9]. 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 [84]. Research has also proven IRL method to generalize well to not earlier encountered settings [65]. Generally speaking, RL often has a lack of generalizability [72]. A solution could be to increase the sample dataset and train on multiple (random) environments [86], 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 [72]. There have been PS algorithms that generalize relatively well. Most of these are episodic-based policy evaluation strategies [76]. Even though some examples can be found in the literature on the 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 as, for example, more assumptions have to be made. Overall, BC methods are quite easy to implement. The reason for this is they are often model-free [9]. 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 is quite costly [8]. A solution for this is to first run the system in simulation. This can, however, not completely replace real-time learning, as even small differences in the environment can result in totally different behaviors.

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 [46]. As RL often uses a continuous stream of data, the amount of data becomes large. This makes it often quite difficult for RL method to deal with robotic learning. Nevertheless, PS has proven to be able to scale well with high-dimensional state space [5]. The same cannot be said for value function methods.

The last criterium relates to *online* learning. For each method, an online implementation can be found. However, these implementations often require 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 [28, 87], 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) [62]. 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 [72]. Instead, RL is in most cases used in an online setting, where the policy is iteratively updated based on the sensory feedback [8].

2.6. Summary

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

- **IL and RL**: Two main robot skill learning approaches have been identified, which both have the benefit that the behavior does not need to be manually programmed. IL uses demonstration, exerted by an expert, to learn skills. Which makes learning easy and natural. In addition, these methods are often computationally efficient as only a small amount (in some cases just one) of iterations 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 a different embodiment of the learner 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 an incorrect reward function can have tremendous results. In section 2.5, a comparison between these two methods has 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 the input state to an action. Most implementations are quite simple and computationally efficient. However, often it only tends to work when a large amount of data is required, due to a compounding error caused by covariant shift (error due to difference in expert distribution and learners distribution). 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, however, at the cost of computational efficiency.
- **RL methods:** Value function tries to find an optimal policy, by iteratively optimizing the value function. A disadvantage of this approach is that it is often difficult to apply in high-dimensional state and action space, which is problematic for robotic applications. 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 actor-critic. 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 is quite limited.

3

Continuous Skill Learning

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

The remaining of this chapter consists of an introduction to different learning outcomes in section 3.1. Next, the effect of skill learning, using different state inputs and actions, will be examined in section 3.2. Section 3.3 is dedicated to methods using a reward/cost function to compute a policy. And lastly, the findings will be summarized in section 3.4.

3.1. Learning Outcome

To learn a certain task, the learning outcome needs to be defined. In [13], three learning outcomes for Imitation Learning (IL) problems were named: a policy, a reward/cost function, and a plan. The choice of the learning outcome depends on the task and the associated constraints. The first two learning outcomes have already been discussed in chapter 2, i.e. the usage of Behavioral Cloning (BC) and Reinforcement Learning (RL) methods to compute a policy and the usage of Inverse Reinforcement Learning (IRL) methods to compute a reward/cost function. The latter, however, requires some additional explanation. This method includes learning at the highest level of task abstraction, where it is assumed that the task is performed according to a structured plan, made up of several sub-tasks of primitive action. These plans typically encode the pattern and constraints in the sub-tasks or primitives and take the robot from a beginning to a goal state. For the remainder of this research, this specific topic will not be discussed separately as these methods, often separate their task into multiple skills which then can be described using a skill's specific policy [88]. Also, the reward/cost function as learning outcome will be excluded from further research, as the literature had proven to be limited when researching real-world implementations using online learning.

Looking at policy as a learning outcome, Ravichandar et al. [13] defined different taxonomies to classify the different methods: policy input, policy output, and policy class. Policy input involves identifying the appropriate input to the policy, an example being a state. The policy output (or action) is similar to the abstraction defined in section 2.2. Policy class refers to the class of mathematical function to which the policy belongs. As the goal of this chapter is to show the continuous learning of robots, using this as a classification might be a bit too abstract as functions do not give a clear idea of how the different implementations hold together. In the remaining of this chapter, the learning of the robot will be compared based both on the state input (type of policy input) and on the action.

3.2. State input/Action

To illustrate how the state input and action connect to the learning of the robot, a schematic illustration was made (fig. 3.1). In the learning scheme of the policy, there are often two segments that make use of some type of input: the policy and the robot. These segments interact with each other through the state input and action. As only online learning will be considered, the control scheme will either be a complete close-loop (as in figure) or it will close-loop with an additional feed-forward loop [89]. In the latter one, the feed-forward loop will, in most cases, consist of one demonstration, this is often the case for approaches using Dynamic Movement Primitive (DMP) [87]. To learn a policy, data is required which describes the state of the robot, therefore referred to as state input. After learning a policy, the robot will be control by means of output of the policy (or action). Both these concepts are used to define how a robot skill changes in the learning process, and will therefore be discussed in this section. In [13], multiple state inputs were identified, where the ones discussed in this section are position, joint angle, and force. Just as for the state inputs, multiple actions can also be identified, with the one examine in this section being impedance, and torque. An overview of the different papers which will be discussed in these sections is given in table 3.1, the state input or action underlined is the base for the discussion about the continuous learning. It should be noted that in this section only papers are included which use some IL method.

3.2.1. Position (state-input)

Position is often used as state input for policy learning using IL methods [13]. There are multiple ways to require this type of data. One method is to use the embedded sensors in the robot, which is especially effective when using kinesthetic teaching to require the data. However, often the data should be required from observing a (human) expert. In these cases, extracting the information can be done using

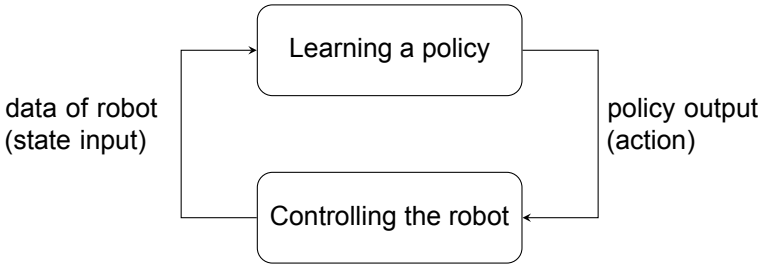


Figure 3.1: Schematic illustration of control scheme. The goal of the learning algorithm is to learn the policy, which can then be used to control the robot in order to execute a certain behavior. The configuration of the robot (e.g. position or force) can be measure and is used by the learning algorithm to find this policy. It should be noted that this control scheme is simplified. In addition, it does not always have to be in closed loop form. However, as this is only meant to illustrate the definition of the state input and action, it is shown like this.

Table 3.1: Overview of papers in which continuous learning is implemented. The method desribes both the encoding and learning method. The demonstration and task which are named in this table represent the relevant information for this chapter, in some cases, other demonstration types and/or tasks were also executed.

Author & year	Ref.	State input / Action	Method	Demonstration	Task
Havoutis et al. (2017)	[90]	Position / Force	TP-HSMM	Kinesthetic teaching	PiH
Calinon et al. (2007)	[44]	Joint Angle / (N/A)	GMM + GMR	Shadowing & Kinesthetic teaching	Gesture imitation
Pastor et al. (2011)	[83]	Force / Position	DMP	Kinesthetic teaching	Grasping
Nemec et al. (2013)	[87]	Force / Position	DMP + ILWR	Teleoperation	PiH
Peternel et al. (2017)	[28]	(Force/position)/ Impedance	DMP + LWR	Shadowing	Sawing
Peternel et al. (2016)	[91]	EMG /Torque	DMP + LWR	Shadowing	Periodic movement.

vision and/or motion sensors [52, 92]. Most state-of-the-art approaches consider vision, as it seems to be the most natural way to capture information from the human. However, these systems must deal with problems such as occlusion, appearance change, complex human-robot kinematic mapping, and the encoding of the video material. Therefore, using a motion sensor can be a more practical approach. This allows for the tracking of the motion preciser and a simple mapping can be established.

An example, where the position was used as the state input, is in [90]. Havoutis and Calinon proposed a framework for supervised teleoperation with online learning. By using an online Bayesian non-parametric learning algorithm, the motion of the robot could be presented online by task-parametrized Hidden Semi-Markov Model (TP-HSMM). These motions can be executed autonomously using a model predictive control (MPC) approach. In the paper, the principle is illustrated using a simplistic example in a planar environment, as visible in fig. 3.2. By means of teleoperation, three demonstrations were done. This result is a model with six Gaussian components. Once it is the remotes robot's turn, it is visible the system is able to create a path to the goal from a new initial position, which means that it is able to generalize. Here, it should be noted that the starting point is indicated by the demonstrator. A graphical representation of this idea has been presented in fig. 3.3. In addition to its generalizability capabilities, also smoothness is visible.

While the planar example gives a clear overview of the main principle of this approach, the eventual goal of the paper [90] was to execute a peg-in-hole (PiH) task. In this experiment, the left arm of a robot is used to kinesthetically demonstrate the behavior, and the right arm to remotely reproduce it. The experimental setup is shown in fig. 3.4. The model is updated online, using five trials. In the reproduction phase, to find the goal position, an RGB-D sensor was used. This showed the ability of the system to generalize well, even when the starting and/or goal position was much different from demonstrated data. A reason why this worked well was the availability of information about the environment. For example, if no RGB-D sensor was used, the new goal positions would not be found, hence it would not generalize well.

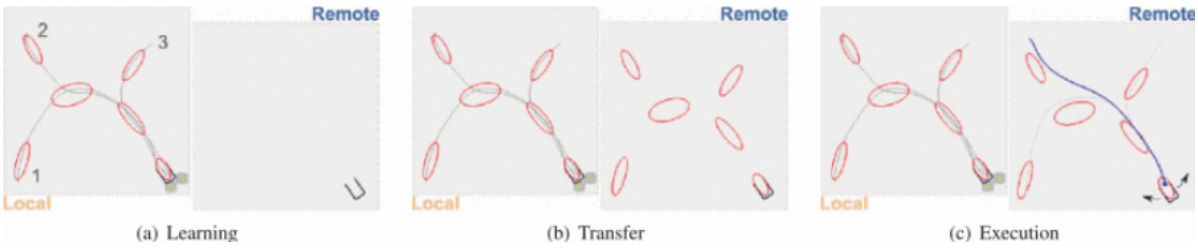


Figure 3.2: Planar example of learning and control with the TP-HSMM [90]. In each subplot, the left side represents the local system (operator's side) and right side the remote system (robot side). The first figure (left) represents three motions and six corresponding Gaussian (here Gaussian are added after each demonstration). The middle figure shows the transfer from the local system to the remote system. And the right figure shows a new motion which is generated by the MPC controller.

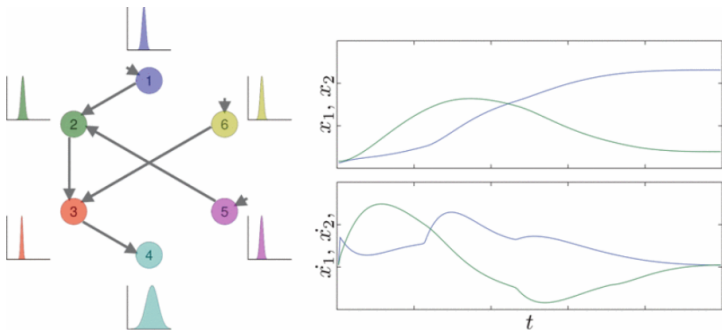


Figure 3.3: Planar example [90] with (left) the graphical representation of the transition matrix and duration probabilities for each state, where the three paths are formed according to the three demonstrations. And (right) the executed motion on the remote side starting from a new position, using TP-HSMM and the MPC controller.

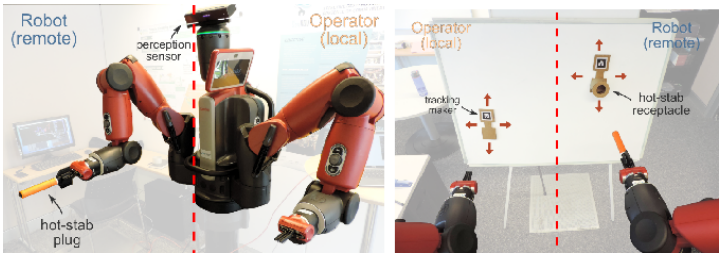


Figure 3.4: Experimental setup of PiH task [90]. The left arm is used by the operator to provide demonstrations and directly teleoperate the right arm. During demonstrations, a TP-HSMM model is learned. This can be used to execute the PiH tasks autonomously using the right arm. Here, the robot locates the new goal positions using an RGB-D sensor.

3.2.2. Joint angles (state-input)

Joint angles are often used to compute the position of an end-effector, making the idea behind using joint as a state input quite similar to using positions. A difference can be noticed in the variability in the configuration. The position is often only used to define the position of the end-effect, leaving some variability in the configuration of the robot. In contrast, the joint angle defines the entire configuration of the robot, thus resulting in a low (sometimes even no) variability. Therefore, the solution space (describing all possible solutions of a task) using joint angles is much smaller compared to using position. While this would result in a more stable system, this also has the disadvantage that it is less able to adapt to changes in the environment and is thus less safe. Another main problem arising with the usage of joint angles as state input is the correspondence problem, as the joints of an expert might not be the same as those of the robot. However, this can be avoided by using kinesthetic teaching for demonstration. For these last cases, the embedded sensors of the robot can be used to obtain the state inputs for the learning algorithms. However, for the other situation (e.g. teleoperation or observation), external sensors are required. Just as for position, vision and motion sensors are a good option for this.

In [44], a wearable device was attached to the human expert, which included eight sensors that each provided its absolute orientation. The goal of this research is to transfer the motion from a human to a (small) robot. This consisted of two steps: an observation step and a kinesthetic teaching step. In this first step, the recorded joint angles are projected in the latent space and encoded using Gaussian Mixture Model (GMM). The training of the parameters is done using Expectation Maximization (EM). In the second step, kinesthetic teaching is used to refine the motion of the robot. The updating of the GMM is done by estimating online the new incoming data and the previous estimation of the GMM parameters. One of the approaches used for updating the GMM is the direct update approach, which uses an adapted version of EM. Results for one of the gestures is shown in fig. 3.5. The usage of both the motion sensor and kinesthetic teaching allows for dealing with the correspondence problem.

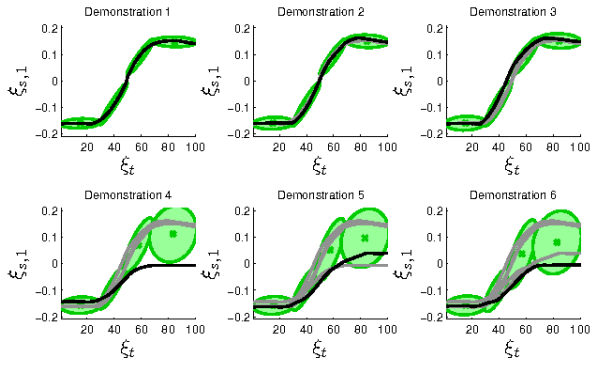


Figure 3.5: Updating of the GMM using the direct update incremental learning approach [44]. The graph shows the encoding of the data in GMM. Each algorithm uses only the latest observed trajectory (illustrated by a black line) and the previous estimation of the GMM parameters, to update the models. Demonstrations 1-3 made use of only the motion sensors, and in demonstration 4-6 the additional information was required using kinesthetic teaching

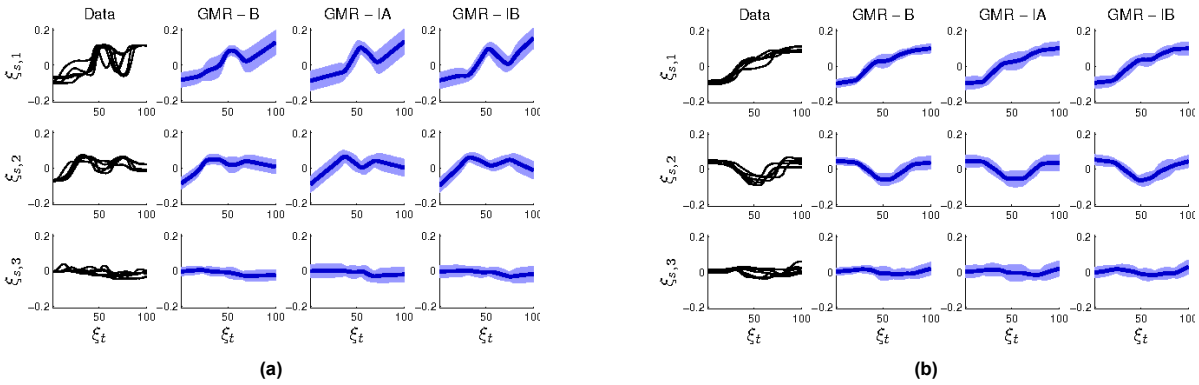


Figure 3.6: Reproduction of different gesture movements using Gaussian Mixture Regression (GMR) in a 3D latent space of motion [44]. *(a & b)* visualize three different gestures. For each gesture three models are trained: one with different batch *B* and two incremental training methods *IA* and *IB* representing the direct update and generative method.

In addition to the direct updating method, a generative method, which uses EM, was also implemented to update the model. Both methods were compared to an offline learning approach: batch learning. For the reproduction of the motion GMR was used. The results for three different gestures are visualized in fig. 3.6. The results show that each method was able to capture the essential characteristic of the demonstrated data. In addition, the results seem to be quite smooth.

3.2.3. Force (state-input)

While position and joint angles can provide a suitable solution in some cases, often they do not account well for adaptation to the environment. The reason for this is that they are simply unaware of the changes. Take for example the grasping of a cup, if the cup is not located at the same position as it has always been and there is no additional feedback, then the robot will simply not grasp the cup. A solution for this is the usage of some force feedback. This case, the robot could feel whether it is touching the cup and adjust its trajectory based on this. This would, however, only work in situations where the difference is quite small, as the next example will indicate.

Pastor et al. used a DMP framework to encode stereotypical movement and extended this for online movement adaptation using sensory feedback [83]. This framework was applied to a grasping task, using DMPs sensory feedback and nonlinear position and force control. The learning scheme used for this is visualized in fig. 3.7. The low-level controller was separated for the end effector (i.e. the hand) and the fingers. As visible in the graph, the used DMP is changed as a result of the demonstrated data

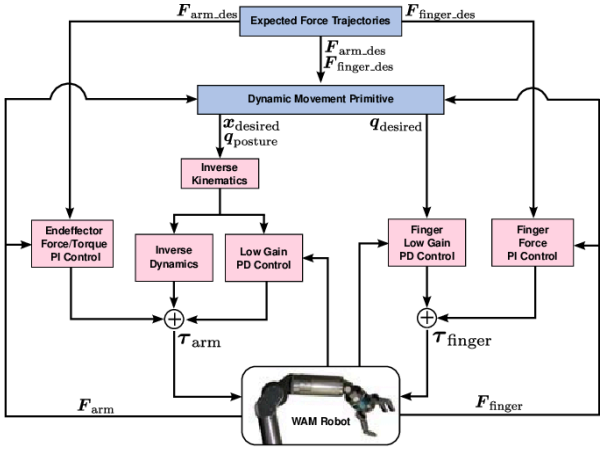


Figure 3.7: Learning scheme for a grasping task [83]. The low-level controller for the end-effector (i.e. hand) and fingers are separated. The control law for the finger τ_{arm} is a combination of a position controller (consisting of a velocity-based operational space controller and an inverse dynamics law and feedback error compensation in joint space) and an end-effector force controller. The control of the fingers τ_{finger} consists of a position PD controller and a force PI controller.

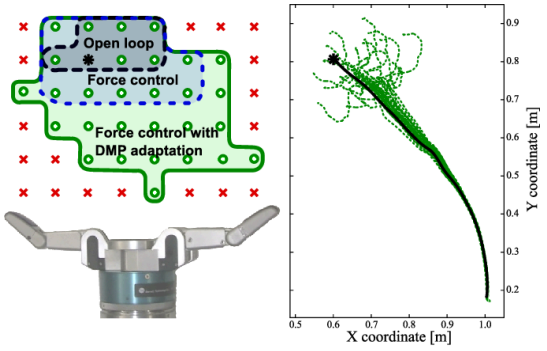


Figure 3.8: Result of grasping attempts after misplacing a flash-light [83]. (left) shows at which positions, the different control modes are still able to grasp the flash-light and (right) shows in black the open-loop trajectory and dashed-green the adapted trajectories after touching the object.

and the previous task execution. In the experiments, the proposed control model was compared to an open-loop and a force control without DMP adaptation control mode. The task for the robot was to grasp a certain object from a table, in the case of the experiment a flash-light. For each control mode, it was tested how they would adapt if the flash-light would be displaced. The results are shown in fig. 3.8. A visible on the left, using DMP with adaptation, significantly increases the area where the robot is still able to grasp the flash-light. In the provided video, it was visible that the robot would always try to go to the position where it ”thinks” the flash-light is located. However, if a force input is provided which indicates that the flash-light is not at the position where it is assumed to be, then it adapts its behavior according to this force feedback. Therefore, it could be concluded that the reason that the robot is not able to grasp the flash-light at any position in the area (as indicated with the red crosses in fig. 3.8), even when the adaptation is used, has something to do with not being in contact with the flash-light at these points (either due to going over it or not reaching it). To deal with this, another form of feedback could also be used, which does not require touching the obstacle. A good example for this is vision, based on which the position of the object could be extracted [93].

Nemec et al. used force as a state-input to learn a policy execute PiH task [87]. The learning scheme has been in fig. 3.9, which shows the main difference, compared to other IL approaches, that not only trajectories but also forces and torques arose during the task demonstration. In addition to encoding the desired trajectory, does the scheme assume that there are still some deviation from the desired behavior, after implementing the learned trajectory. These are a result of inaccurate pose estimation and can be accounted for using some vision feedback. Therefore, the goal of the learning scheme is to reduce this error as much as possible.

To test the proposed scheme (fig. 3.9), two types of robots were used. However, as both obtained similar results, only one will be discussed. The robot, a six Degrees of Freedom (DoF) UR5 equipped with gripper and a force/torque wrist sensor, uses a high gain non-compliant controller and therefore allows for the implementation of an admittance force control law. The robot is also equipped with a magnetic tracker to track the peg’s pose and orientation. The demonstration was exerted using teleoperation with a pose tracker. Three PiH task were exerted using the robot, of which the results shown in fig. 3.10 are of the second task, where the robot has to insert a square peg when the baseplate has been rotated and translated. As expected, iteration results in a lower offset of the position, however, as visible, the offset is not completely removed. This does not necessarily have to be an issue, as multiple trajectories can be used to exert a similar task. Looking at the force, it is visible that the force used in the learned trajectories is lower than the demonstrated one. The paper names as reason an imperfect force sensor calibration, this is most likely to be accurate, as the goal is to minimize the error between the demonstrated and generated forces. Therefore, directly trying to recreate the demonstration thus leads to suboptimal performance. It was also noticed that the robot would try to replicate the learned force/torques rather than the position/orientation of the human demonstration. This, however, does not necessarily have to be a problem, as there can be multiple trajectories that result in the execution of the task.

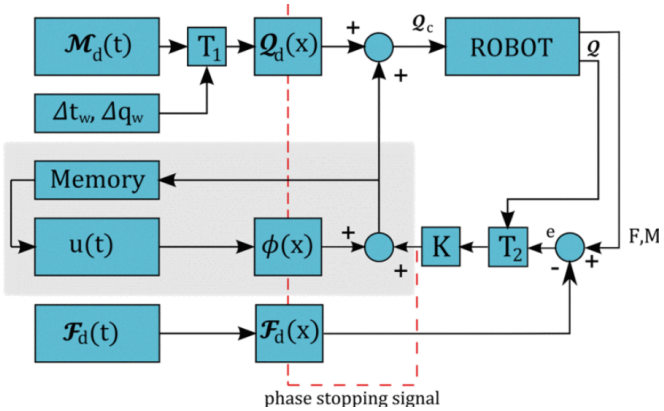
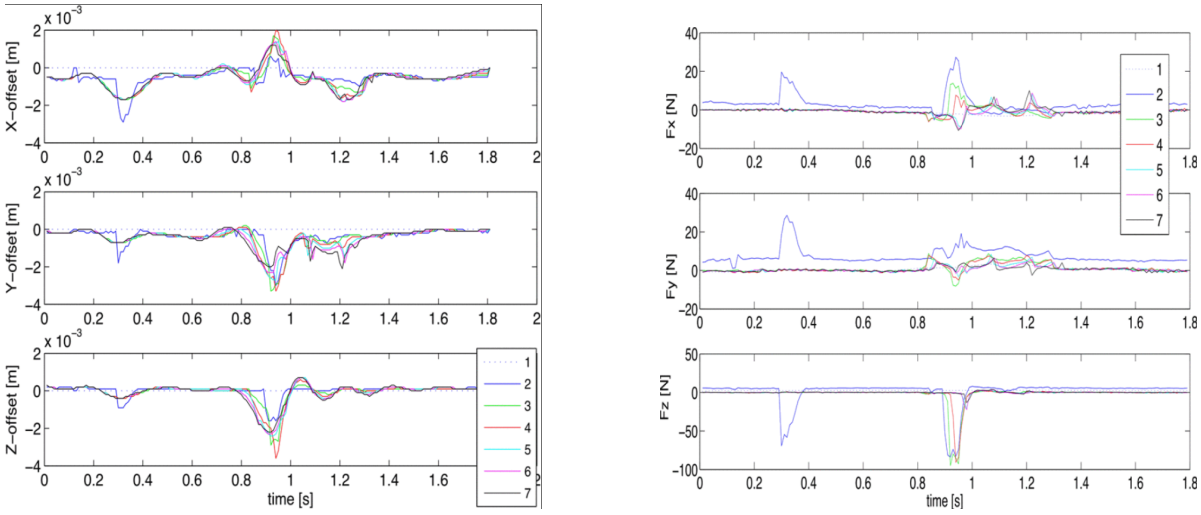


Figure 3.9: Learning scheme of PiH tasks [87], describing an iteratively updated DMP using both position/orientation and force/torque input, where $\mathcal{M}_d(t)$ denoted the original demonstrated trajectory encoded by DMPs, Q_t the output signal obtained by integrating the transformation equations of the DMPs and applying workpiece displacement (Δt_w, Δq_w) as estimated by a vision sensor (this takes place is block T_1), $\mathcal{F}_d$ denotes the captured forces and torques captured during demonstration they are encoded using a linear combination of Radial Basis Function (RBF), the eventual goal is to minimize the difference between the measured force/torque and the desired force/torque (denoted as e), in T_2 this error is transformed into robot base coordinates. By using admittance control, the position/orientation offset is calculated and added to the existing offset of the previous iteration step $\phi(x)$. The motor commands Q_c are then a combination of the motor commands obtained from the DMPs (Q_t) and the offset. The iteration procedure is repeated until the measured forces, match the desired forces.



(a) (b)

Figure 3.10: Results of PiH task with a rotated and translated baseplate [87]. *(a)* The learned positional offset of 6 consecutive cycles and demonstrated trajectory denoted by 1, which has no offset. *(b)* The force measured during the execution of the task in 6 consecutive cycles, again 1 denoting the demonstrated trajectory.

3.2.4. Impedance (action)

In [28], the goal was to learn an impedance scheme to control a robot while performing a collaborative task (a sawing task is used in the paper). A big difference, compared to the earlier showed experiments, is that instead of learning from demonstration exerted by a human expert, the robots would learn from other robots. The proposed learning scheme for this research is presented in fig. 3.11. To allow for robot-to-robot learning, a novice robot (robot without skill) first had to be taught by a human expert. This consisted of two steps: observing the human to obtain some basic skill, and learning the sawing task through collaboration. This is different from conventional IL implementation, where the robot learned by mimicking the behavior of the human. After learning, the robot, who is now assumed to be an expert, can be used to learn other novice robots the same task. The interesting part of this research therefore lies in the red indicated part (fig. 3.11), which consist of three stages: a learning stage where the novice robot is completely compliant and the expert robot has high stiffness, a stage where the

novice robot learns the desired impedance behavior which results in a a follow/leader role in different phases of the task, and a third stage where both robots are assumed to be an expert. DMP was used to encode the trajectories in the first stage, and Locally Weighted Regression (LWR) to learn these encoded trajectories.

In fig. 3.12 the learning process of the novice robot is shown, where the interesting part occurs in the third stage. In theory, the stiffness pattern of both robots should be the same, except for a phase shift of half a period (due to switching between pulling the saw toward itself and being compliant when it is pulled away). As visible, the stiffness pattern of the novice robot is not at all the same as the pattern of the expert robot. In the previously shown experiment [44, 83, 87, 94], the aim was always to mimic the human expert as good as possible, however, as this experiment uses collaboration between the robots to learn the skill, randomness could occur. A theory could be that the robot has, due to this randomness, improved its skill in comparison to the expert robot. However, to determining whether this is the case, further research should be conducted. A disadvantage of randomness is the possibility that behavior becomes worse, therefore making it less reliable.

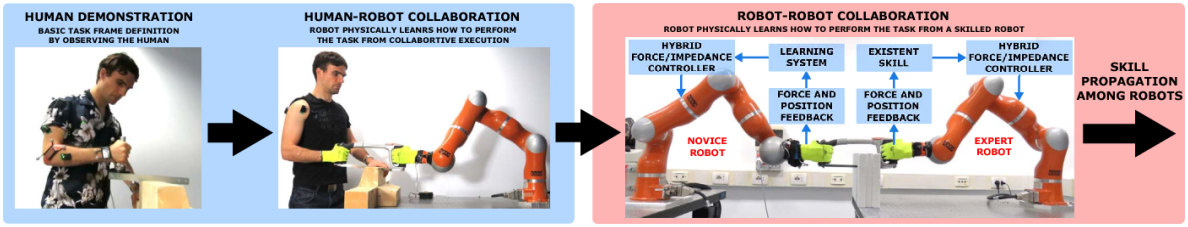


Figure 3.11: The skill learning of a novice robot [28]. The blue area illustrates the skill acquisition for an expert robot, based on the concept presented in [95]. This starts with demonstrations from the human which is observed by the robot, this allows for defining the task frame control framework. Next, the expert robot uses this knowledge to collaborate with the human and learn the task specifically, using physical interaction. The next area (indicated with red) illustrates the proposed method in the paper, where the novice robot learns from the expert robot. This part can be propagated among other robots.

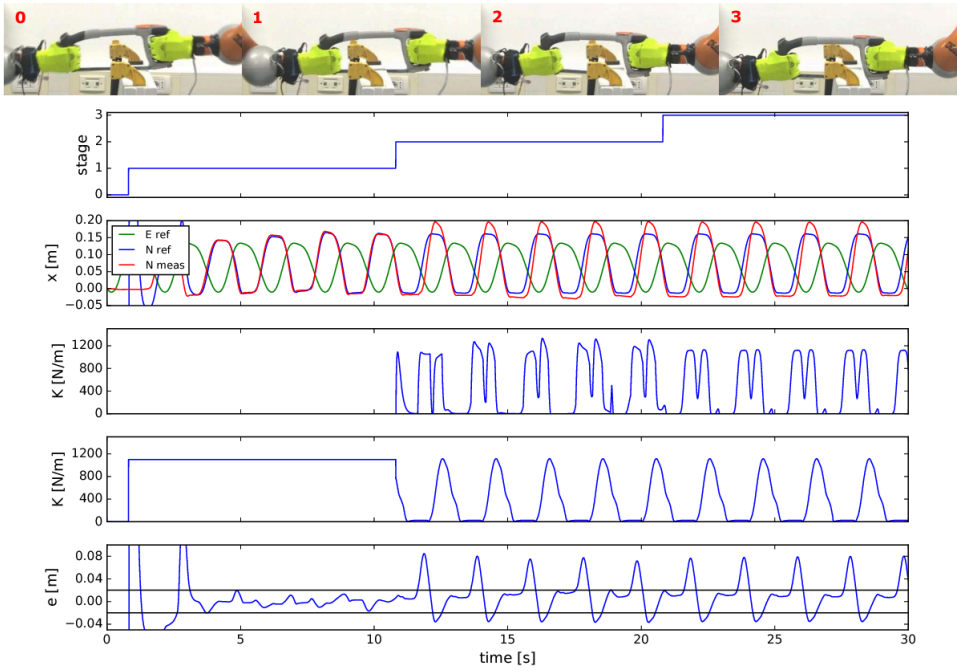


Figure 3.12: Results of robot-robot collaboration experiment on two-person sawing task [28]. The sequence of photos on top shows the sawing progress throughout the different stages. Next, from top to bottom, the graphs visualize; The different learning stages. The sawing motion in the x-axis, where the green line represents the reference motion of the expert robot, the blue line the reference motion of the novice robot, and the red line the actual motion. The stiffness of the novice robot. The stiffness of the expert robot. And lastly, the error between the reference and the actual motion used in the stiffness learning process.

3.2.5. Torques (action)

In [91], an exoskeleton is used to reduce the effort exerted by an expert while performing a periodic task. This was done by learning a torque trajectory, which should be exerted by the robot to alleviate the human, therefore making torque the action. To learn this, a scheme was presented (fig. 3.13). The Biofeedback, measured using electromyography (EMG), is used to estimate the phase of the signal, and as a state input for the learning of the trajectory. For the learning a framework with DMP to encode the trajectory, and LWR to learn it, was implemented.

In the paper, different experiments were conducted, each having the goal to reduce the human muscular effort as much as possible. In the tasks, the human subject had to move an object repetitively from one position to another at a certain frequency. While doing so, the exoskeleton would try to adapt to the human and, by doing so, reduce the effort. One of the experiments used an elbow exoskeleton, meaning that the subject would only be supported by torque exerted on this specific joint. The subject had to execute three different phases, where each phase consisted of a target elbow joint range and frequency. The results are visible in fig. 3.14. The third plot shows muscle activity. As visible, each phase first consists of the subject putting in much effort until it reaches the desired behavior. Once it researches this behavior, the effort is reduced to zero due to the adjusted torque (visible in second plot).

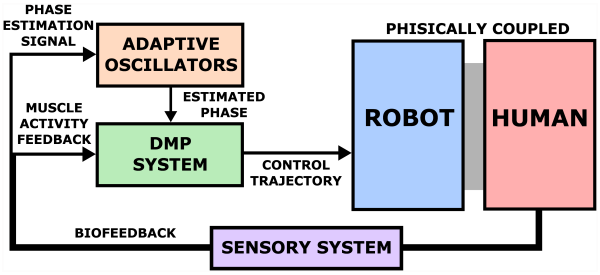


Figure 3.13: The control scheme of an exoskeleton as presented in [91]. The human is included in the control loop, by using sensory biofeedback obtained with EMG. The goal is to minimize the human muscle activity feedback by learning an appropriate assistive torque behavior in the joint. The Adaptive oscillator extracts the phase and frequency information from the human biofeedback and adjusts the phase and frequency of the learned feed-forward trajectories based on this. Based on this estimated phase and the muscle activity feedback, a DMP is created, which results in a control trajectory.

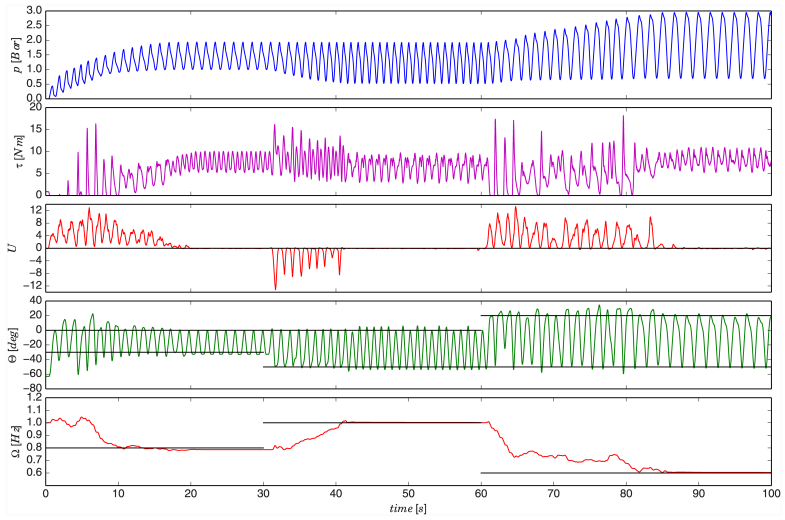


Figure 3.14: Results of the experiment on elbow exoskeleton using a trained subject [91]. From top to bottom the different plots illustrate the: command PAM pressure learned by the DMP system p, the joint torque calculated based on the force measurement of the load cell τ, the human muscle activity based on feedback obtained with the measured EMG U, the angle of the robot elbow joint Θ and the frequency of the motion as extracted by the adaptive oscillator from the muscle activity feedback Ω. The reference states are indicated by a black line.

3.3. Reward/Cost function

RL has the advanced that the learned skill is not limited by the knowledge of the demonstrator. However, it has the limitation that it requires the reward or cost function to be known. The choosing of this function is critical as the resulting behavior depends on this. If this function is not known, using IRL could be an option. However, this again requires human demonstration, thus adding the drawback of being limited by its skill. In this section, the continuous learning of a robot using a reward or cost function is shown. The skill improvement will be analyzed based on the improvement in the received reward or reduced cost. In the first part, section 3.3.1, this will be done using a regular RL approach. In addition, an implementation of RL using human advice is described in section 3.3.2. None of the discussed papers have used IRL to find their reward function, while being promising, implementing IRL for skill learning has proven to be difficult and is therefore not often done in research.

3.3.1. Regular learning

In [96], actor-critic RL (described in section 2.3.4) was implemented to execute a trajectory following task. In the paper, two control methods were introduced: a RL-based reference compensation and a RL-based input compensation. The former is formulated as a function of the joint space error and its derivative. The latter directly compensates for the reference signal fed to the system. Two of the tracking task, used to test both RL approaches, will be discussed below. For these tasks, a UR5 robot, controlled using velocity commands, is used.

The first task is to follow a discontinuous trajectory, namely a square. The goal is here to reduce the error in the z-axis. The video provided by the paper shows the learning process of the robot. The first iterations show that the robot is a bit higher above the table than required. It also shows an overshoot when trying to follow the squared shape trajectory. It is noticeable that when the square-shaped trajectory starts, there is nearly no delay in trying to follow it. During the learning process, the overshoot becomes incrementally better. The results also show that the skill improvement is the highest at the start of the learning process, and slowly convergence until the average best performance is obtained. The learning process based on the sum of rewards (return) is visualized in fig. 3.15. For the input-based method, the return of each agent (joint), slowly converges to an average best solution. In contrast, the reference-based method shows erratic behavior. The paper stated two possible reasons for this; Firstly, it can be caused by the RBFs which are initialized with an inappropriate value, causing the learning curve to deteriorate before improving. And secondly, the reference discontinuities introduce very large errors, which cause the policy value function to first behave erratically before converging to stable behavior.

The second task is to follow a continuous trajectory, namely a circle. The goal is to minimize both the x-axis and y-axis errors. The provided video shows that initially, the robot is already able to follow the circle trajectory quite well. Looking at the two different methods, fig. 3.16, it is visible that using the input-based compensation result in much faster convergence. This is a result of directly reducing the

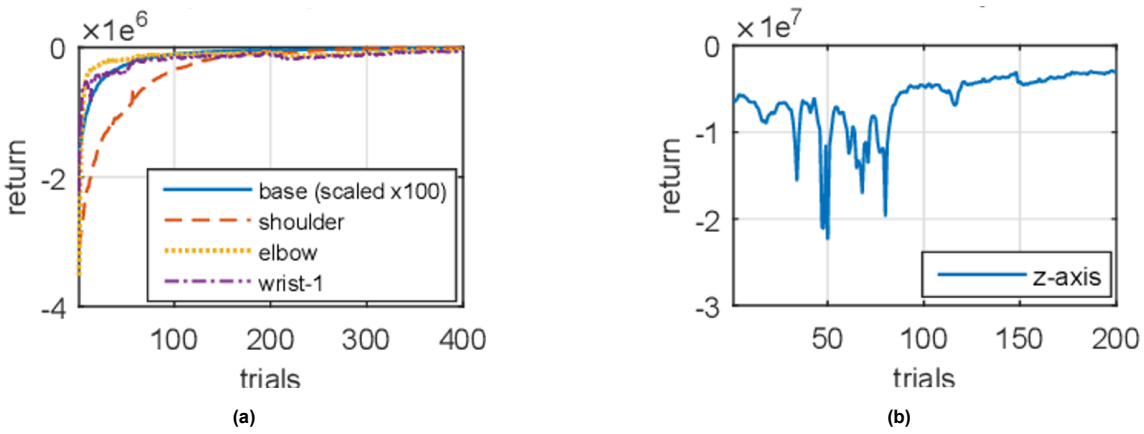


Figure 3.15: The sum of reward (return) for the squared trajectory following [96]. With *(a)* the RL-based input compensation, and *(b)* the RL-based reference compensation.

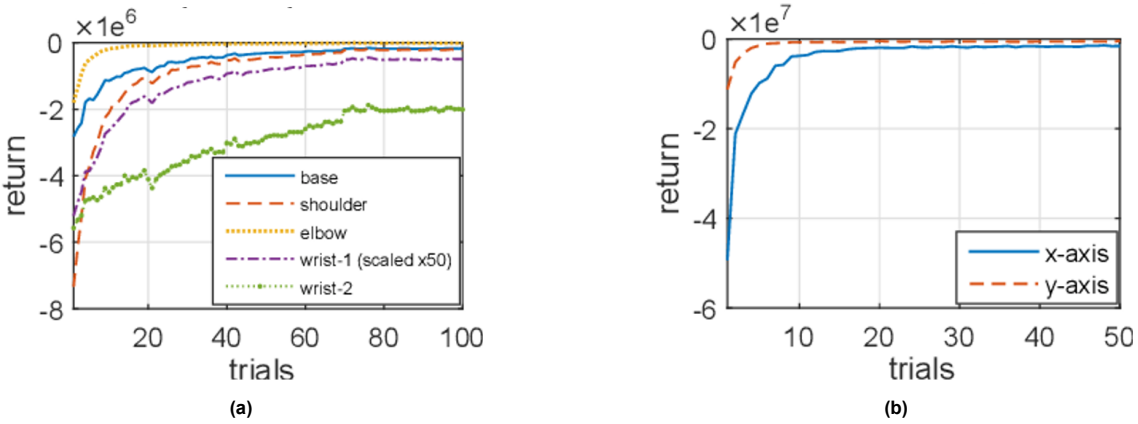


Figure 3.16: The sum of reward (return) for the circle trajectory following [96]. With *(a)* the RL-based input compensation, and *(b)* the RL-based reference compensation.

trajectory error, which means it directly compensates for the joint velocity. The response of this method is however a bit slower, as the corrected trajectory is tracked using a nominal PD controller. Lastly, as this method modifies the reference instead of the control input, the result is slightly smoother than that of the input-based controller.

In [96], multiple reasons are stated why either one of the control functions would be better for certain criteria (response time, convergence speed, smoothness, etc.). Besides these criteria, the importance of choosing a suitable control function for a specific task can be seen. Where the reference-based compensation does not even work for the following of the discontinuous trajectory, it results in better behavior for the continuous trajectory.

3.3.2. Human Advice

In section 2.4, the usefulness of combining IL and RL was already expanded on. This idea was implemented Celemin et al. [47], where human feedback was used to excel the policy beyond the capabilities of a human expert. In their research, two methods using feedback were implemented. Firstly, an adapted version of Corrective Advice Communicated by Humans (COACH) was used, which allowed the incorporation of the teacher correction in the policy search (PS) loop. During execution, the human teacher can give occasionally advice, to correct the behavior. Secondly, they proposed *interactive PS*, where human guidance was enabled based on COACH to influence and bias the exploration of a PS algorithm in the form of exploration noise.

To examine the performance of the approaches, a *ball-in-cup* task was used, as shown in fig. 3.17. Here the reward function was separated into two parts: one where the height of the ball is lower than the cup and one where it is higher. This is an extension of the earlier used approach [97], as that reward function would not provide any useful information if the ball was lower than the cup. Due to the

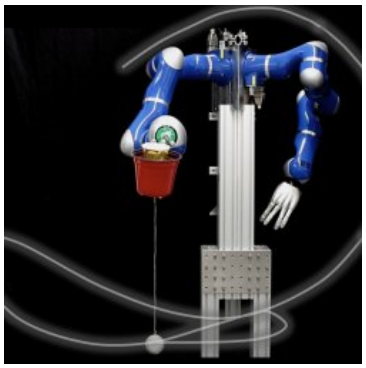


Figure 3.17: Snapshot of ball-in-cup experiment [47].

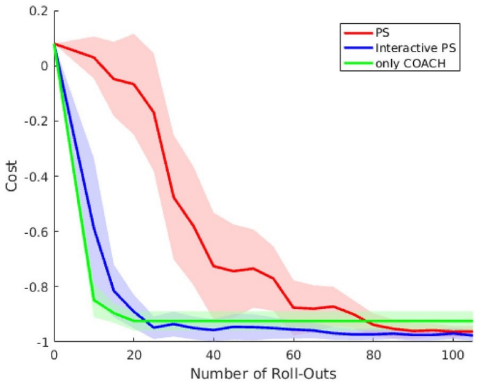


Figure 3.18: Convergence curve of *ball-in-cup* when learning from an initial demonstration [47]. This task was learned, using a standard PS method, an interactive PS method, and PS with COACH.

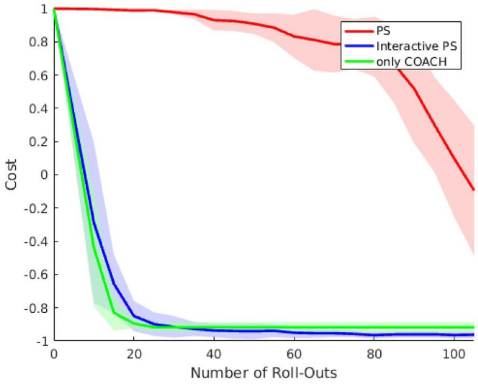


Figure 3.19: Convergence curve of *ball-in-cup* when learning from scratch [47]. This task was learned, using a standard PS method, an interactive PS method, and PS with COACH.

extension, the task can be considered as two sub-tasks: *swinging the ball* and *catching the ball*. The policy was represented using Probabilistic Movement Primitive (ProMP) and PI^2 was the base for the PS algorithm.

The first demonstration used a policy derived using kinesthetic teaching as an initial starting point. The learning curve of this task has been presented in fig. 3.18. It is visible that both methods using human feedback converge initially much faster than the standard PS method. COACH and the interactive PS are able to achieve policies that can catch the ball at 10 and 15 episodes, respectively. Whereas the original method has this at 60. Noticeable is that the COACH implementation does not improve much after the initial catch, which can be explained as the human will stop giving advice once the ball has been caught, either because the need for improvement is not evident or it is not necessary. This shows the inability of a human to improve once they are near an optimal policy. Controversy, the interactive PS learns a bit slower at the start, which is probably due to some earliest roll-outs in the update process, however, it keeps improving until it researches the best average performance around 70 episodes. The pure PS, might converge slower, but reaches, just as the interactive PS method, a better performance than the solely COACH method. These results show that the advantages of combining IL with RL is that it also combines its features: faster convergence (due to IL) and researching of a better performance (due to RL).

In the second set of experiments, the relevance of choosing a good initial policy is shown. In contrast, to the earlier experiment, does this one start from scratch. The results are visualized in fig. 3.19. The effect of starting from scratch is visible as the starting point is now at 1 instead of 0.1. This has especially a big effect on the PS method, which takes a lot longer to converge now, showing its sensitivity to the initial policy. At the start, the learning process is quite slow as the random movement seems to reduce the effect of the previous ones. The results do not even show the PS receiving a behavior where it is able to catch the ball, however, if more episodes were conducted, it is reasonable to assume that it would eventually be able to. Looking at the interactive PS and the COACH method. A similar convergence behavior is visible as in fig. 3.18. While it takes them both a bit longer to catch the ball, they eventually achieve similar performance as before.

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

3.4. Summary

The goal of this chapter was to illustrate how skills can change during the learning process. The main findings of this chapter are:

- **Learning outcome:** There are multiple types of learning outcomes, the two elaborated on are: a policy, and a reward or cost function. For online learning, the research for the latter has proven to be limited. Policy learning can be classified based on multiple aspect, but the two used in this

chapter are state input and action (policy output). The state inputs examined are position, joint angle, and force, and the actions examined are force and impedance.

- **Position and joint angles as state-input:** The benefits of position and joint angle often are quite similar. This is logical, as the position is often a translation of joint angle. While both these inputs have proven to result in good performances, they also seem to be limited when relating to adapting to environments. Changes in the environment cannot be easily adapted to by just using the configuration of the robot. Instead, some types of additional feedback need to be provided in these cases, examples are vision or some sensory information.
- **Force as state input:** The advantage of using this state input, is their ability to adapt to small changes in the environment. After slightly shifting an object from its original position, if the robot still touches it, it would be able to successfully grasp the object. However, just as for position and joint angles, additional information is required to find poses totally different from the original.
- **Impedance and torque as action:** Both these methods showed the ability of the system to adapt to a desired behavior. Using action to examine the behavior has proven to be more difficult than using state input. A reason for this could be that if something in the control changes, you want to know why, and this can only be explained when looking at the state input, which then can also be referenced to as the feedback.
- **Collaborative task:** These types of tasks are promising for future research. Task using some kind of demonstration (teleoperation, kinesthetic teaching, etc.) try to find a behavior that mimics the behavior of the expert as good as possible. Whereas this is also the original goal of the collaborative task, research shows mutated behavior which can - but does not necessarily have to be - better than those of the expert.
- **Reward/cost function:** If set up correctly, using reward/cost function to compute a policy, has the potential of finding a (good) solution after multiple iterations. However, choosing the learning system is critical, as improper definitions can result in slow convergence or even no convergence at all. In addition, the choosing of the initial policy can have large results in the computational time. A method to compute a good initial policy is by using IL.

4

Discussion

This literature report has aimed to give an overview of methods that could be used for continuous skill learning. In this chapter, the main finding will be discussed with the aim to relate the found skill learning methods, identified in chapter 2, to the effect of skill learning discussed in chapter 3. In this discussion, two main topics will be discussed: the limitations of the robotic learning methods, and robot generalizability capabilities.

4.1. Limitation of robotic learning methods

In chapter 2, multiple methods have been identified which can enable continuous skill learning, the two most broad being Imitation Learning (IL) and Reinforcement Learning (RL), which both come with their advantage and disadvantage. Starting with IL, this approach allows for a reliable and safe design, which often results in good generalization capabilities [7, 9, 65]. A major drawback of this method, is the fact that they are limited by the skill of the expert. In the literature, three solutions that show potential to tackle this problem have been identified.

- **Collaborative task:** Often in IL, the robot is trying to mimic the behavior of the expert as good as possible. This concept was expanded on in [28], where instead of imitating the expert, a collaborative task resulted in the robot gaining a skill. While it might not be preferable to let the robot completely learn from scratch, using a collaborative approach, could be complementary to a basic skill obtained by imitation. For example, the robot would require a basic skill by observing an expert. After which, it collaboratively performs a task. Whether the idea would work mainly depends on the type of task which should be learned. For example, the collaborative sawing task in [28] can be learned to use such an approach, whereas it might become more difficult for the peg-in-hole (PiH) task in [90]. In addition, it might not be desired to execute a task collaboratively in the execution phase (takes place after the learning phase, when the robot autonomously performs a task). Therefore, an idea would be to first learn the policy using a collaborative task, after which this should be translated to a solely performed task, so it can be used in the execution phase.
- **Noise in data:** Another option would be to allow for a lot of noise in the data [87]. In essence, this results in more randomness in the solution and thus has the potential of becoming better. However, a major drawback of this idea is that it also has the potential of becoming much worse. In addition, this method can not provide any stability guarantees, which potentially results in an unsafe system.
- **Combine with RL:** A last promising solution, is to make RL complementary to IL. Here, IL would be used to learn an initial policy in a low amount of iterations, after which the skill can be improved using RL. An example of this was given in [47], which showed the improvement in computational time, compared to using a RL approach, when using IL to determine an initial policy and RL to improve this.

The idea behind each of these methods is to search a larger part of the state-action space will be examined to obtain good behavior. The searching of a large part of this space is typically done by RL.

Two major drawbacks of this approach are, however, their computational time (as many iterations are needed to compute a solution), and the limitation of the skill by the choice of reward function [8]. This first drawback was already discussed, as combining with IL would increasingly reduce this time. The choice of reward function is however an ongoing topic. While most research mentions the importance of choosing a correct reward function, the effect of choosing it incorrectly seems to be limited. Instead, most implementation has either based their function on previous research [47] or they neglected to mention why this would be the best function in comparison to any other possibility [96]. Even though this function often does not seem to be having the best reasoning behind it, it often still provides a solution for the given problems. This results in the question of what a good solution is. Some literature has used the completion of a task as a reward function. However, this binary approach often does not seem to be the best option, as not finishing a task always results in no reward, while there can be a big difference between just not finishing the task, and not even coming close. Therefore, reward shaping might be useful [80], which is a method for providing more frequent feedback on the appropriate behaviors. A method for finding rewards is Inverse Reinforcement Learning (IRL) showing to close link between RL and IL methods. This again results in the limitation of IL to only be as good as the expert.

4.2. Generalizability

One criterion, discussed in section 2.5 to compare the different skill learning methods on, is generalizability, which is an ongoing topic in robotic learning [67]. Generalizability describes the ability of a system to deal with settings/environments which were not uncounted during the learning phase. As previously discussed, some examples of generalizing are: being able to deal with states and action which are unobserved in the learning phase, starting the task at a different initial state, and reusing skills in different problem settings [5]. It was found that some methods showed more potential to generalize than others, e.g. IL methods often generalize better than RL methods. In the papers discussed in chapter 3, some points relating to generalizability came forward.

Starting, most papers only assume the ability to start at a different position or to go to a different position, when talking about generalizability. Some examples given are [83, 87, 91, 94], where a distinction between large and small changes could be made. In [83], the system was able to grasp an object standing at a different position, while some could say that this method thus had the capabilities to generalize, I beg to differ. Instead, it could be stated that this system had the ability to adapt by using feedback. Initially, it would move towards the position at which the object used to be placed, if during execution, force feedback was felt, the robot would adapt its movement such that it would still be able to grasp the object. However, if the object was not touched, it would also not be able to grasp it. While this ability to adapt to small changes can definitely be beneficial, e.g. when the position of an object can not precisely be determined, it does not mean it is able to generalize.

Another example is given in [87], where a PiH task was performed. After using demonstrations to train the model, sensory feedback was used to detect the position where the peg had to be inserted. Using this sensory information, a position could be extracted. Compared to the earlier discussed research [83], does this show some generalization capabilities, as the system is able to deal with all different positions due to the feedback received. However, again it is limited as the task (to insert a peg) stayed the same. If, for example, the task changed into pulling out the peg instead of inserting it, the system might not be able to do this with the learned set of skills.

Concluding, while some methods have been stated in literature to generalize better than other methods, it was found, that in order to generalize well, it is necessary to have information about the environment. It should be mentioned, that while the usage of sensors to extract information about the world (based on which generalization and also adaptation could be done) is frequently implemented, it has not thoroughly been examined in this research. Therefore, additional research should be acquired to make better conclusions on this subject. Additionally, in most research found, only the ability of to move from or to a new position was discussed. It could, however, also be desired for the system to execute its skill in a totally different environment or to use the learned skill for a different task.

5

Conclusion & Future research

In this chapter, the research question, defined in chapter 1, will be answered. This is followed by some advice for further work.

5.1. Conclusion

The research question which will be answered in is as following:

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

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

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

In literature, it was found that both approaches have other skill learning processes. Starting with Imitation Learning (IL), different imitation methods allow for different learning processes. The main distinction can be made between methods using one demonstration and methods using multiple. For the one demonstration, the learner can either learn a policy in one go, which cannot be seen as an online implementation, or the one demonstration can be used to compare the behavior of the robot to (as done in [87]). In this last approach, the policy is iteratively updated until the difference between the demonstrated data and the behavior of the robot is minimized. This thus assumes that the optimal behavior is not learned in one go. For multiple demonstrations, the learner can either obtain all demonstrations at once and compute a distribution based on this, which is again not an online implementation, or each demonstration is used to iteratively update the policy. On option for the latter is to take during each new iteration only the last generated policy into account, instead of using all the previously obtained data [28, 44]. By doing so, the system would be less computational expensive, as the amount of data used tom compute the new policy is much lower.

RL often enables online learning, as each iteration the policy is updated. The skill learning shows that with each iteration, the behavior becomes slightly better. Often the system starts with updating into one direction in the state-action space (the one direction meaning towards an optimal/good behavior), unit ill it overshoots. After the overshoot, the direction is shifted, resulting in a zigzag behavior around the optimal solution (this optimal solution can either be global or local and depends on the choosing of the reward function). An example is the ball-in-cup experiment [47], where it first undershoots a couple of times, next it overshoots, followed by an undershot, and so on until it eventually goes through it.

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

5.2. Future research

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

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

While reward functions are often mentioned as being highly critical for the result of the RL algorithm, comparing different functions for robotic learning has not been found. Therefore, it could be useful to compare different reward functions for the same task and examine which effect this has on the learned skill.

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

Combining the features of IL and RL methods into one approach could result in the robot exerting the behavior of using either of these approaches singular. While some research has already shown the potential of this idea, there is a lack of comparison between combining these approaches and using them singularly. Therefore, this could be an interesting research topic.

References

[1] Sylvain Calinon et al. “A probabilistic approach based on dynamical systems to learn and reproduce gestures by imitation”. In: *IEEE Robotics and Automation Magazine* 17.2 (2010), pp. 44–54.

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

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

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

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

[6] Shixiang Gu et al. “Continuous deep q-learning with model-based acceleration”. In: *International conference on machine learning*. PMLR. 2016, pp. 2829–2838.

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

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

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

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

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

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

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

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

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

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

[17] Stéphane Ross and Drew Bagnell. “Efficient reductions for imitation learning”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 661–668.

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

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

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

[21] Maria Kyrarini et al. “Robot learning of industrial assembly task via human demonstrations”. In: *Autonomous Robots* 43.1 (2019), pp. 239–257.

[22] Olivier Sigaud, Camille Salaün, and Vincent Padois. “On-line regression algorithms for learning mechanical models of robots: a survey”. In: *Robotics and Autonomous Systems* 59.12 (2011), pp. 1115–1129.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[70] Richard S Sutton and Andrew G Barto. “Reinforcement Learning: An Introduction”. In: (1998).

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

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

[73] David Silver. *Lecture 4: Model-free prediction*. 2015.

[74] Miguel Morales. “Grokking deep reinforcement learning”. In: Manning Publications, 2020. Chap. 6.

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

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

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

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

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

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

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

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

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

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

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

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

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

[88] George Konidaris et al. “Robot learning from demonstration by constructing skill trees”. In: *The International Journal of Robotics Research* 31.3 (2012), pp. 360–375.

[89] Mariantonieta Gutierrez Soto and Hojjat Adeli. “Recent advances in control algorithms for smart structures and machines”. In: *Expert Systems* 34.2 (2017), e12205.

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

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

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

[93] David A Duque, Flavio A Prieto, and Jose G Hoyos. “Trajectory generation for robotic assembly operations using learning by demonstration”. In: *Robotics and Computer-Integrated Manufacturing* 57 (2019), pp. 292–302.

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

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

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

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