University of Paderborn

Learning and imitation in heterogeneous robot groups

Wilhelm Richert

Dissertation
in Computer Science

submitted to the

Faculty of Electrical Engineering, Computer Science, and Mathematics

in partial fulfillment of the requirements for the degree of

doctor rerum naturalium (Dr. rer. nat.)

Paderborn, 2009

Supervisors:
Prof. Dr. Franz J. Rammig, University of Paderborn
Prof. Dr. Hans Kleine Büning, University of Paderborn
Prof. Dr. Uwe Brinkschulte, University of Frankfurt

Date of public examination: 22. December 2009

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dr. Franz Rammig for his guidance and constructive feedback. I also thank Prof. Dr. Hans Kleine Büning and Prof. Dr. Uwe Brinkschulte for vice-supervising my thesis, as well as Prof. Dr. Friedhelm Meyer auf der Heide, Prof. Dr. Achim Rettberg and Dr. Matthias Fischer for reviewing my work.

This work would not have been possible without the many fruitful discussions with and suggestions from my group leader Dr. Bernd Kleinjohann and my colleagues Dr. Lisa Kleinjohann, Dr. Dirk Stichling, Dr. Christian Reimann, Mr. Markus Koch, Claudius Stern, Philipp Adelt, and Andreas Thuy. I am especially grateful to Dr. Natascha Esau for her guidance in expressing my ideas as concise mathematical formulas.

Neither would this work have been possible without the bright students I had the opportunity to work with and who contributed to this work: Oliver Niehörster, Raphael Golombek, Ulrich Scheller, and Riccardo Tornese. I also wish to thank Marina Scheiderbauer and my brother Johann Richert for revising the English of my manuscript.

Last but not least, I heartily thank my wife Natalie for her patience and support as well as our two little sons Linus and Moritz for constantly reminding me of things in life that are way bigger than two letters in front of a name will ever be.

Abstract

As robots become increasingly affordable, they are used in ever more diverse areas in order to perform increasingly complex tasks. These tasks are typically preprogrammed by a human expert. In some cases, however, this is not feasible – either because of the inherent complexity of the task itself or due to the dynamics of the environment. The only possibility then is to let the robot learn the task by itself. This learning process usually involves a long training period in which the robot experiments with its surroundings in order to learn the desired behavior. If robots have to learn a shared goal in a group, the robots should imitate each other in order to reduce their individual learning time. The question how this can be done in a robot group has been considered in this thesis, i. e., how robots in a group can *learn* to achieve their shared goal and *imitate* each other in order to increase the performance and the speed of learning by spreading the learned knowledge in the group.

To allow for this intertwined learning and imitation, a dedicated robot architecture has been developed. On the one hand, it fosters autonomous and self-exploratory learning. On the other hand, it allows for manipulating the learned knowledge and behavior to account for new knowledge gathered by the imitation process. Learning of behavior is achieved by separately learning at two levels of abstraction. At the higher level, the strategy is learned as a mapping from abstract states to symbolic actions. At the lower level, the symbolic actions are grounded autonomously by learned low-level actions.

The approaches of imitation presented in this thesis are unique in that they relieve the requirements that governed multi-robot imitation so far. It enables robots in a robot group to imitate each other in a non-obtrusive manner. The robots can thus increase their learning speed and thereby the overall performance of the group by simply observing the other group members without requiring them to stick to a certain communication protocol that would provide the necessary information. With the presented approach, a robot is able to infer the behavior that the observed demonstrator is performing and to replay the beneficial behavior with its own capabilities.

In addition, the presented approaches allow the robots to apply imitation even if the group is heterogeneous. Normally, the performance of a group degrades if robots with incompatible capabilities imitate each other. Capability differences arise if robot morphologies differ in a robot group. This is the case if different robots from different manufacturers form a robot group that has to achieve shared goals. This thesis presents an approach that is able to determine similarities or differences between robots. This can guide the robots in a heterogeneous robot group in order to determine those robots for imitation that are most similar to themselves.

Contents

CHAPTER 1

Introduction

By three methods we may learn wisdom: First, by reflection, which is noblest; Second, by imitation, which is easiest; and third by experience, which is the bitterest.
Confucius, Chinese philosopher

Of the methods by which we can gather wisdom or knowledge, *imitation* is often considered as an inferior shortcut to the more creative "noble" or "bitter" ones. The *imitator* is thereby contrasted as dumb or lazy against the creative and eager *imitatee*. Yet, imitation is one of the most powerful means to spread learned knowledge. With imitation, the imitator is relieved from individual exploration, which leads to a drastic speedup of the learning process. This thesis explores approaches that allow imitation to be combined with individual learning in heterogeneous robot groups. It will be shown, how the learning speed of the robot group can be increased and thus the self-organization of the group can be supported.

As a matter of fact, imitation plays an important role in the development of humans (Fig. 1.1). They are able to imitate at an age as early as 12 days [123]. Being such a powerful means of knowledge acquisition, imitation also has been observed in animals [50]. The imitation incidents show significant differences in quality, though. There is, e. g., the more intelligent version of imitation – often found in humans – that tries to analyze and interpret the imitatee's actions, in order to infer their original purpose. The other side of the spectrum shows a much simpler imitation type, called mimicry, which tries to copy only the actions or appearance of the imitatee. Independent of the sophistication level of imitation, it obviously pays off in nature.

For the above reasons, imitation has already been widely adopted in robotics research (cf. Chap. 2). The possibility to let robots in a group benefit from each other's experience not only speeds up the learning phase, which is essential in today's complex robots. It also decreases wear out and damage, which is often involved in the exploration process.

Figure 1.1: Humans are capable of imitation at an early age

When trying to provide robots with imitation capabilities, one is faced with three challenges corresponding to the three steps involved in imitation [44]:

- *Recognition:* Salient bits of the observed behavior have to be extracted from the raw observation.
- *Transformation:* The recognized complex behavior has to be transformed from the perspective of the imitatee into a data structure that is comprehensible for the imitator.
- *Generation:* New behavior has to be generated from the properly encoded data.

Current research often focuses on one of these challenges, requiring everything else to be specified by hand – mostly in a context where a human is the imitatee and the robot has to reproduce the observed task [28, 51, 60]. Attempts that employ imitation in a multi-robot context combining learning and imitation so far still require important challenges to be solved by the human expert beforehand, such as the actuator mapping between the imitator and the imitatee or even the possibility to look into the other robot's internal data structures [142, 175].

What is still missing, is a truly autonomous multi-robot imitation approach. That is an imitation approach that does not require human intervention to solve any of the imitation-specific challenges. In this case, the following requirements have to be met:

- The imitation approach has to rely only on subjective information perceived directly by robot's sensors.
- A robot has to decide autonomously when it is imitating and when it is learning individually.
- A robot has to decide autonomously what to imitate and how to integrate the observed behavior into its own behavior knowledge.

As the robot group shall be capable of learning to achieve its goals without human intervention, the robots must be able to learn individually. Therefore, the above listed requirements can only be met if imitation is intertwined with learning.

1.1 Objectives and contributions

This thesis considers the question of how robots can be designed so that they

- *learn* to achieve their shared goals in a robot group and
- *imitate* each other to increase the learning speed by spreading the gathered knowledge in the group.

In order to allow for this intertwined learning and imitation, a dedicated robot software architecture is vital, which on the one hand fosters autonomous and self-exploratory learning while on the other hand allows for manipulating the learned knowledge and behavior to account for new information retrieved during the imitation process. Therefore, the imitation part will need information regarding the low-level behavior, which interacts directly with the environment, to detect and classify observations. The learning part will have to integrate the recognized behavior in the component containing more abstract high-level behavior. Hence, the architecture has to support this at multiple levels of behavior abstraction. Based on this architecture with imitation support, the thesis will develop algorithms for autonomous robots that result in robust system behavior and spread the system behavior refinements to other members of the system group.

The approach shall be evaluated in so-called Capture-The-Flag scenarios, in which items are scattered in the environment to be collected by robots in a group and delivered to one or more goal bases. This mimics the typical natural scenario of *prey retrieval*. Prey retrieval is being performed in many natural systems and often used as a model for a range of other real-world applications, as dirt cleanup, search and rescue, or searching of terrain samples in unknown environments [107]. It is one of the canonical tasks for multi-robot systems [57, 114] and comprises the major difficulties the robots will face when deployed in the real-world.

In order to show that imitation combined with individual learning has provided benefits over learning-only approaches, experiments easy to conduct like those often performed in simple real-world robot scenarios are no longer possible. When imitation is combined with individual learning, it is no longer possible to count a specific behavior as a result of imitation. The robot might have individually learned the specific behavior as well. Instead, a multitude of experiment runs have to be performed with only learning enabled and with imitation and learning both working together. Only this kind of experiment is able to show significantly, whether the imitation together with individual learning has provided any benefits in terms of learning speed over the learning-only experiments. Since a magnitude of long-running experiments are needed to show statistically significant results, the approaches are evaluated in the physically realistic 3D simulator *Player/Gazebo* [79]. For the dynamics, Gazebo relies on the *Open Dynamics Engine (ODE)* [168], which is the current standard for the type of robot experiments [46, 69] needed to evaluate the approaches in this thesis. In addition to already existing robot models that are

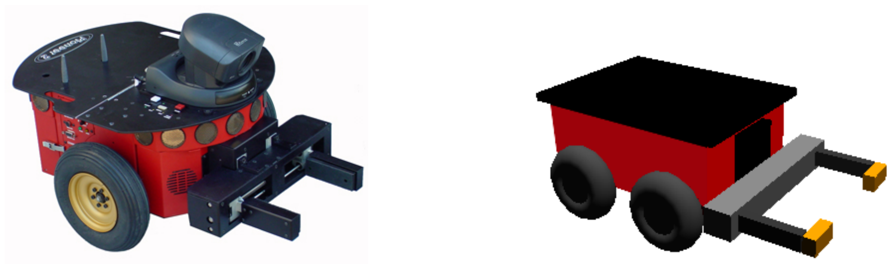


Figure 1.2: Real Pioneer robot and simulated Pioneer robot basis [25]

very similar to their real-world counterparts, Gazebo offers support for easy modification and extension of those models. As the base robot platform the Pioneer robot class is used [25], which already exists as a module for the Gazebo simulation environment (Fig. 1.2).

Besides the novel architecture offering support for imitation, the thesis presents unique approaches relieving the constraints that governed multi-robot imitation so far, to allow for the following:

- **Robots in a robot group with shared goals can imitate each other in a non-obtrusive manner.** They can thus increase the learning speed and the overall group's performance by simply observing the other group members without requiring them to stick to a certain communication protocol that would provide necessary information. With a technical realization of the concept of the mirror neuron system found in humans and animals, the robots are able to infer which behavior the observed demonstrator is performing. With the presented techniques, the imitator is then able to replay the beneficial behavior with its own capabilities.

- **The robots in a group can be heterogeneous.** Normally, the performance of a group degrades if robots with incompatible capabilities imitate each other. Capability differences arise if the robot morphologies differ in a robot group. This is the case if different robots from different manufacturers form a robot group that has to achieve a shared goal. This thesis presents an approach that is able to determine similarities or differences between robots. This can guide a robot in a heterogeneous robot group to determine those robots that are most similar to itself.

1.2 Thesis outline

After providing the background for the two prevailing topics of learning and imitation, this thesis presents the architecture that is necessary for the two approaches concerning imitation, namely imitation in robot groups and the choice of the imitatee. The detailed chapter outline is as follows:

Chapter 2 (*Background*) starts with the three major types of learning and continues with presenting and contrasting different imitation notations that have been added to the scientific language over the past century. Due to the huge body of research that exists in both research fields, this chapter coarsely delineates both fields so that forthcoming chapters can locate their specific research contributions accordingly.

Chapter 3 (*Architecture for learning and imitating in groups*) provides an overview of the architecture consisting of the three layers *motivation*, *strategy*, and *skill layer* and highlights their interaction. The chapter then explains how the layers are used for imitation in robot groups and for the choice of the imitatee.

Chapter 4 (*Motivation layer*) defines the goals of a robot by means of a motivation system that is able to serve as an internal reward. The robot uses this motivation system to express its current motivation as an overall well-being state to other robots.

Chapter 5 (*Strategy layer*) presents the strategy learning and state abstraction mechanisms that are needed to autonomously learn to achieve the goal defined by the motivation layer.

Chapter 6 (*Skill layer*) explains how the robot can explore its own capabilities and learn low-level actions, called skills, that can further be used by the strategy layer.

Chapter 7 (*An integrative example*) shows exemplary how the three layers are used to build a fully autonomous robot that can learn to collect objects in the environment and carry them to a goal base.

Chapter 8 (*Imitation in robot groups*) presents the algorithm, which uses the strategy layer and skill layer to detect complex behavior in observations and integrates it into its own learned behavior knowledge.

Chapter 9 (*Choice of the imitatee*) handles the question how the imitation approach can be employed in heterogeneous robot groups, where the correct choice of the imitatee is vital. The chapter describes how Bayesian networks on interaction capabilities are used to achieve a measure of similarity between two robots.

Chapter 10 (*Summary and outlook*) concludes this thesis by summarizing its main points and providing suggestions for future improvements.

CHAPTER 2

State of the art

This thesis describes how a layered learning architecture for robots is used to support self-organization in robot groups by combining learning and imitation. While the concept of *learning* is well-defined in the literature, this is not the case with the term *imitation*. As a consequence, it has no single agreed definition and is misused in a great body of research. At the one side of the spectrum, it is referred to as some kind of dull copying of behavior that was ingeniously developed by another subject. The *imitator* is thereby contrasted as dumb and lazy against the creative and eager *imitatee*. At the other side, imitation is recognized for the cognitive capacity necessary to intelligently imitate another subject.

This chapter provides a broader view on both topics, learning and imitation. It is not meant as a complete overview, though. The purpose is to coarsely delineate both fields of research so that forthcoming chapters can locate their specific contributions.

2.1 Learning

Current learning approaches can be grouped into *supervised*, *unsupervised*, and *reward-based* learning. This section describes their application domain, strengths and weaknesses.

2.1.1 Supervised

Supervised learning methods require a permanent critic that provides the correct actions for the given state while the robots are in the learning phase. They are therefore used often in a single-robot setup, where the robot is taught by a guide to perform the desired behavior [171].

Supervised learning methods are usually not applicable in multi-robot scenarios because of the complexity of interaction in the robot group. A critique would otherwise have to provide all participating robots permanently with an individual feedback regarding their specific last action.

2.1.2 Unsupervised

Unsupervised learning methods are used in special cases like learning models to support robot navigation [103]. Unsupervised methods do not require any external feedback. The only source of information is the environmental data without any meta information regarding the outcome of the recent behavior. *Clustering* is one example of unsupervised learning. Based only on the relationship of a given data set the clustering approach groups the objects into similar clusters. Another example is the *self-organizing map* [104], a type of artificial neural network that is trained for dimensionality reduction purposes. The problem of *blind source separation* [24] has also to be counted to the unsupervised learning methods. From a set of mixed signals one has to extract the original signals without the support of any further information regarding the sources of information.

2.1.3 Reward-based

A large body of research investigates reward-based methods, which can be divided into the camps of *reinforcement learning* (RL) [173] and *stochastic search methods*, e. g. evolutionary computation [70], simulated annealing [111], or stochastic hill-climbing [77] that directly learn behaviors and do not try to approximate value functions. Within the stochastic search method community of the multi-agent domain, the focus lies on evolutionary computation. It is a well-known fact that evolutionary methods require many populations. They are usually learned in a simulation context, whereupon the learned behavior is applied to real-world scenarios. This is no option for self-organizing robot groups: In this case all the adaptation and learning has to be carried out in the real world. Therefore, this thesis focuses on the most successful reward-based learning method, which is RL. A more detailed overview of the reinforcement learning approach that is used in this thesis will be given in Sec. 5.1.

2.2 Imitation

Before surveying the body of research that already exists in the field of imitation and with its connection to learning, this section has to map out the landscape of existing imitation concepts and definitions. Subsequently, biological organisms, which are capable of imitation are presented together with a classification of their respective type of imitation. It follows a survey of current imitation research in robotics, which is evaluated and contrasted to the approach of this thesis.

In the following, *organism* refers to a living being, while *robot* refers to the context of this thesis' application. *Subject* will be used if the described context is applicable to both nature and multi robot systems.

2.2.1 Biological background

Imitation has been investigated now for over 100 years. In the nineteenth century, Darwin, as well as the biologist Romanes [152], classified many animal techniques of behavior acquisition as imitation. They did not try to establish a definition of imitation, though. A little later, Baldwin defined imitation in his evolution theories to be any adaptive process [35]. The first crisp definition was then given by the psychologist Thorndike as "learning to do an act from seeing it done" [178].

The definition range of imitation has always been a matter of debate since then. Thorpe suggested a stringent definition of imitation restricting it to only those processes that exactly copy the observed behavior [179]. In favor of a more tolerant definition, Whiten and Ham argued that the definition should focus primarily on the outcome of an imitation process [187]: as long as an animal has learned something useful from another one, it should count as imitation and not require an exact reproduction of the observed action. Between these two extremal points, numerous additional definitions have been placed in the meantime.

2.2.1.1 Categorizing imitation

Recognizing the problem of confusing the definitions of imitation and how this impedes the communication within the robotics research community, Call and Carpenter [54] distinguish *imitation* from *emulation* and *mimicry*. Although they focus on the classification of imitation instead of its technical realization, their taxonomy helps to discriminate between the different types in the body of research that will be described later. They propose a multidimensional framework for classifying an imitation process, which incorporates three different sources of information an imitator uses during the imitation process (Fig. 2.1). These dimensions are

- whether or not the imitatee's *goal* is understood,
- whether or not the observed *action* is copied, and
- whether or not the observed final *result* can be reproduced.

Given these three dimensions, any type of imitation process can be categorized clearly. Sometimes, it is difficult or impossible to determine one or more of these dimensions by only observing an imitation process, however.

This taxonomy classifies an imitation as *mimicry*, *emulation*, or *imitation*, which Call and Carpenter define as follows.

Mimicry is the copying of an observed behavior that superficially resembles the behavior of the observed organism. During mimicry the imitator does not understand the goal, but simply tries to copy the observed action.

Emulation There are two different types of emulation dependent on whether the imitator understood the goal while imitating or not.

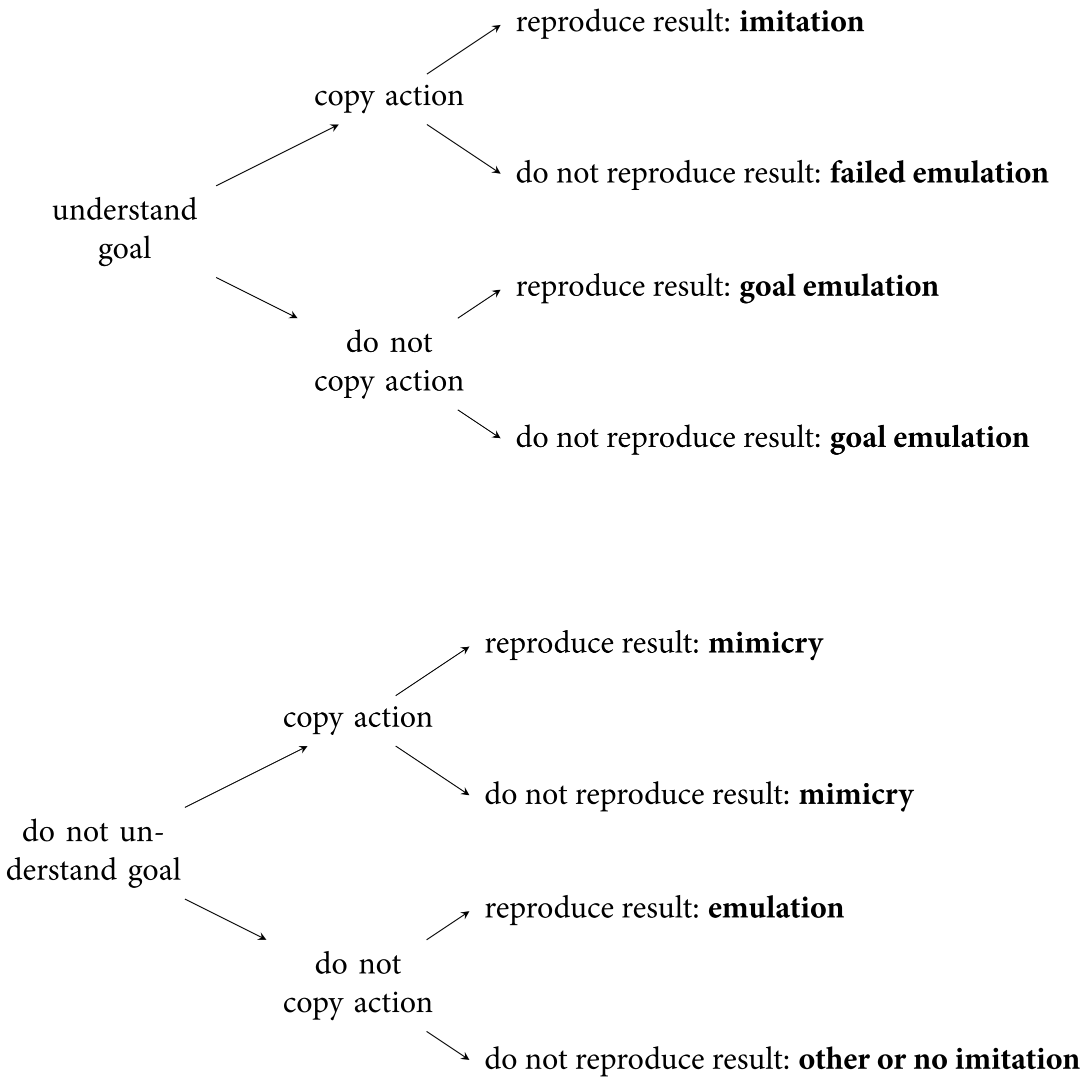

reproduce result: **imitation**
copy action
do not reproduce result: **failed emulation**
understand goal
reproduce result: **goal emulation**
do not copy action
do not reproduce result: **goal emulation**
reproduce result: **mimicry**
copy action
do not reproduce result: **mimicry**
do not understand goal
reproduce result: **emulation**
do not copy action
do not reproduce result: **other or no imitation**

Figure 2.1: The three sources of imitation by Call and Carpenter [54]: *goals*, *actions*, and *results*

Action emulation focuses on the results (changes in the environment) during imitation[1]. It is not of interest whether or not the goal intended by the imitatee is reached. Thereby, the subject may learn how to achieve a change in the environment, but fail to arrive at the goal intended by the imitatee.

Goal emulation focuses on copying the goal without copying the action. This involves creativity on the imitator's side.

Often, it is not easy, and sometimes even impossible, to distinguish between both variants of emulation. E. g., it is not yet resolved whether apes focus on results during imitation [55, 56], which would mean that they emulate, or whether they concentrate on goals [187], meaning that they are involved in goal emulation.

[1]Call and Carpenter called this simply "emulation".

Imitation Whereas *emulation* is based on reproducing the results of an observed action, *imitation* also involves copying the actions that led to that result. An illustration of Call and Carpenter [54] highlights the difference: When an imitatee uses a hammer to open a nut, the emulation of it could also be to smash the nut on the wall or biting the nut to open it. Imitation, in contrast, would involve the hammering action to open the nut.

Comparing the definitions of all the types of imitation, *mimicry* is certainly the least useful in multi-robot applications. Not understanding the goal, mimicry involves a lot of trial and error until the observed result can be reproduced. And if the environment or other conditions change, it is unlikely that the behavior knowledge collected in the previous mimicry process can be transferred to the new situation.

Just as brittle as mimicry, is simple *emulation*. Without understanding the goal, the imitator is missing important information that would help to transform previously learned behavior to new situations that need new environmental changes to achieve the original goal. Even more so, as emulation does not even incorporate segments of the observed action.

Goal emulation and *imitation*, finally, offer much more robustness and promise to reproduce the observed result much faster. Whether or not the observed action is copied seems to be of minor importance, as long as the goal is understood. This is also the imitation type of the multi-robot imitation approach described in Chap. 8. There, the proposed approach involves to understand the imitatee's goal and thereby increases the overall robustness of the imitation process.

2.2.1.2 Imitation and memetics

The imitation process of copying successful behavior between organisms has been investigated by Susan Blackmore ("The Meme Machine" [44]) and Richard Aunger ("The Electric Meme" [30]) from a sociological point of view. Based on Richard Dawkins' work "The Selfish Gene" [65], they analyzed an additional replicator besides the gene, which Dawkins called *meme*. In their view, a meme is any unit of information that can be passed on from one subject to another. With this broad definition they analyzed different kinds of imitation and were successful in describing social learning incidents involving imitation. From the robotics research point of view, however, they could not place memetics on a scientific basis, which allowed to quantify, control, or predict memetics phenomena. The remainder of this thesis will therefore use the notion of imitation.

2.2.1.3 Imitation in biology

As might be obvious, humans are able to imitate. Children are capable of using all three types of information (goal, action, result) [122, 130, 38]. Imitation of facial gestures in its simplest form, mimicry, could be observed at infants of an age as early as 12 days [123].

Available evidence suggests that animals can imitate as well. Young birds, e. g., learn their regional dialect of bird song by imitating the songs of their more mature peers [36, 90]. Kinnaman observed a rhesus monkey (*Macacus rhesus*) that pulled a plug from a box containing food after it has observed another monkey doing so [98]. In the animal kingdom, dolphins are one of the most proficient nonhuman imitators. Kuczaj and Yeater found evidence that dolphins are even capable of delayed imitation [106]. This requires the ability to mentally represent observed behavior. This enables dolphins to even imitating themselves.

Even more surprising, imitation can be observed in fauna [50]. Orchids, e. g., are able to imitate. *Tongue-orchids* developed flowers, which resemble nests that are confused by insects with their natural nests – a form called *visual imitation* in the biology research community. Other orchids, like *Orchis papilionacea* or *Ophrys fusca*, imitate the scents or pheromonal signals of certain insect species. This allures these insects to attempt copulation, which results in flower pollination. A closer look, however, reveals that the imitation attempts of this kind are clearly a form of mimicry and not true imitation in the sense of Call and Carpenter.

2.2.2 Imitation in robotics

Imitation applied to technical systems can be found under diverse terms highlighting the different goals of the use of imitation. The two directions of imitation research most relevant to this thesis are *Programming by demonstration* and *multi-robot imitation*. The former's goal is to program a robot in a more intuitive way – without having to rely on traditional programming languages. This is useful in the case that either the time to program a robot would take too long, or if the task cannot be expressed by the human expert at all. The goal of multi-robot imitation is to let the robots in a robot group benefit from each other's learning efforts. No human guiding is given at runtime in this case.

Before presenting the work that is most relevant to this thesis' focus, the current challenges are described that govern robotics' imitation research today.

2.2.2.1 Challenges in robot imitation

The research in robotic imitation has been following a common theme over the years, which deals with the five central questions and the correspondence problem.

The five big questions in imitation The challenges governing successful imitation in multi-robot systems have been summarized by Dautenhahn and Nehaniv as the "Big Five" central questions in imitation, "namely whom, when, what, and how to imitate, in addition to the question of what makes a successful imitation" [64].

whom In a group of robots, which robot can be regarded as a good teacher or imitatee?

when It is not wise to imitate all the time. The imitator should carefully take into account the situation context of the available imitatees and whether the potential imitatee is in exploration or exploitation mode.

what The imitator should be clear about what to imitate. This comprises at the broad view Call and Carpenter's previously mentioned three sources of information (results, actions, goals; Sec. 2.2.1.1), but also the level of behavior (state sequences, low-level actions).

how How should the observed behavior be mapped into the imitator's own behavior repertoire? This is called the *correspondence problem* and will be treated separately in the next section.

how to evaluate What should be counted as successful imitation?

Although the "whom" is the first of the challenges, it has been factored out in current research so far – either by restricting the imitation process to a one-to-one imitatee-imitator relationship where the roles of both are clear, or by providing the robots with fixed rules. However, the question of whom to imitate plays an important role already in early childhood. This has been shown, e. g., by the psychologist Burnstein [48], who found out that children imitate peers more often that have similar sex, age, or interests.

The "when" has also not been thoroughly investigated up to date. Often, even the starting and ending point of the action to imitate are given in advance. The full action in the given time interval is then to be copied. In multi-robot applications, this is not possible, of course.

The question "what" to imitate is the best explored question in robotics imitation. It led to numerous successful approaches applicable to different levels of behavior. In multi-robot scenarios – the targeted application domain of this thesis – the "what" is restricted to sequences of higher level behavior (state-action-state traces). The information in a robot group is too sparse for imitating low-level actions.

Many approaches in current research take the easy way out when answering the "how" question. Using the tabula rasa approach, the imitator is starting from scratch without any behavior knowledge. In this case, it does not have to cope with combining individually learned behavior with behavior collected by an imitation process. This is, however, the standard case in normal applications.

The last question regarding the evaluation of the imitation success is naturally very specific to the application scenario. Research focusing on the creativity aspect usually employs evolutionary approaches, where the imitation success is measured in terms of behavior emergence. This is not the focal point of this thesis, as it is targeted towards online imitation, where the imitation success should increase within the same robot generation at runtime. Other researchers consider imitation as successful that exactly copies the imitatee's actions. This is also not of relevance for this thesis. Instead, it measures imitation success in terms of how much it was able to speed up the learning efforts.

The correspondence problem The correspondence problem arises from the differences between the imitator's and imitatee's morphology [132, 26]. This is the case, if the imitatee performs an action of which the imitator has no direct corresponding actor. If, e. g., a robot with four wheels tries to imitate a human walking on two feet, the robot has to interpret the observed action (walking to a goal position) and find the corresponding behavior that results in the same effect (control commands for the individual wheels).

In nature, the correspondence problem is solved by the *mirror neuron system* in the brain of humans and primates. It contains a special class of neurons, called *mirror neurons*, which were discovered first by Rizzolatti and Craighero in the macaque monkey premotor cortex [150]. Mirror neurons fire both when an organism performs a particular action and when it observes the same action performed by another organism. This indicates that behavior recognition is tightly coupled to the process of behavior generation. Inamura et al. argue, this fact even suggests that both processes are realized as a combined information processing scheme [94]. There is evidence for mirroring as a more general principle [85], which applies to tactile sensation [97] and to emotions such as disgust [188] and pain [164].

In case of technical systems, solving the correspondence problem is a hard problem. In order to successfully solve the corresponding problem, the imitator has to understand the goal of the imitatee. Consequently, a subject that is able to solve the correspondence problem is also capable of not only mimicry but also of true imitation or goal emulation in the sense of Call and Carpenter (Sec. 2.2.1.1). Johnson and Demiris found out that the correspondence problem can be solved by focusing on the features of the demonstration that are important to the imitator [95]. This involves an action abstraction mechanism.

2.2.2.2 Programming by demonstration

Programming by demonstration (PbD) tries to relieve the robot programmer from the tedious low-level programming task (a manipulative task or gesture, like grasping an object, e. g.) by demonstrating the task in question repeatedly to the robot [51]. PbD is sometimes also referred to as *programming by example*, *learning from demonstration* [28], *apprenticeship learning* [60], *behavioral cloning* [156, 96] or *scaffolding and moulding* [157], a form of self-imitation, in which a human expert remotely controls a robot performing progressively complex task. Programming by demonstration is preferably done by guiding the robot. Thereby, the human teacher moves the robot's components (arms or legs in the case of a humanoid robot) to perform the desired action [53]. The robot then has to learn from the recorded sensori-motor data to copy the presented task. Goal understanding – and often the final result of the task – are often not relevant; the presented task just has to be copied. This is a form of mimicry and often used in research on human-robot interaction. PbD is therefore best applied in situations, where human experts know how to perform the task, but not necessarily know how to express their performance. With PbD, robots could be programmed for restricted environments to cook [89], to forage [78], to control a helicopter [59], and to play tennis [92] and air hockey [41].

If the teacher is the human being, special techniques of action recognition are necessary to extract salient information from the observation stream of the human performance. The recognition of human action has therefore attained much attention spanning approaches based on hidden Markov models (HMM) [189, 29], support vector machines [133], Gaussian processes [163] or conditional random fields [167]. Perera et al. apply a Multi Factor Tensor model to recognize styles and person identities in human movement sequences [138]. Their approach needs a physical model specified beforehand and relies on a motion-capturing system to extract the motion data. Nejigane et al. use boosting to robustly recognize online human motion data [134].

Also for the reproduction of recognized behavior diverse techniques have been used. Some are based on HMM and use the Viterbi algorithm to synthesize behavior thereof. Billard et al. [43, 52], as well as Azad et al. [31], e. g., use the Viterbi algorithm to let the upper part of a humanoid robot replay behaviors observed at a human expert. In their work, the imitator-imitatee roles are known and fixed. Also the start and end points of the behavior to imitate is known by the robot. They split the imitation task into the observation and imitation processes, having the goal to minimize the discrepancy between the demonstrated and imitated data sets. Lee and Nakamura even showed how a robot can imitate observed human behavior it if the robot only saw parts of the human body [110]. They used HMM to encode the observed behavior and Viterbi to reproduce it afterwards.

In all these approaches, the robot is only able to learn low-level behavior and this can only be done from scratch. They are not suitable for behavior learning in robot groups, since not only a human teacher is missing in the multi-robot case, but also the close imitator-imitatee relationship is not given any more.

2.2.2.3 Imitation in multi-robot systems

Although the prevalent approaches in the PbD domain are able to demonstrate successful usage of imitation in technical domains, they all suffer from requiring the imitator-imitatee relationship to be fixed, with the imitatee often being a human. Furthermore, the time frame in the observation, to which the imitator has to pay attention, often needs to be provided beforehand. The task to be learned by imitation is then repeated several times and afterwards the robot has to derive a generalized representation of the imitated task and be able to replay it.

Only a few current approaches, which have the need to use imitation in multi-robot systems, address these questions. A survey on how they do so in the field of robotic soccer [175], computer games [142], and general robotics [94] will be given in Sec. 8.1. The survey will also highlight, how the sporadic imitation approach, which will be proposed in this thesis, improves upon them.

In multi-robot systems where the robots are allowed to imitate, the question naturally arises whom a robot willing to imitate should choose for the imitation process. The previously surveyed approaches ignore this issue by treating only homogeneous robot groups. For application in heterogeneous groups, a means that measures the capability differences between the group members is needed.

Up to now there is very limited research addressing this issue. While some require very detailed information of the robot's inner states and actions [33], others require the correspondence problem to be manually solved [162]. How these approaches work in detail, and how the provided solution of this thesis stands out in that it allows the application of imitation with only minimal predefined information also in heterogeneous robot groups, will be considered separately in Sec. 9.1.

2.2.3 Contrasting the thesis to the state of the art approaches

In contrast to the existing approaches, this thesis combines the capability of imitation with layered learning architecture of a robot to be used in realistic multi-robot scenarios. It does so in a way that allows the approach to be used without requiring the other robots to reveal any internal state or executed action except for the overall well-being state. In addition, the approach presented in this thesis does not require all robots to use the proposed architecture. Robots that do, will benefit. This enables a group of such robots to increase their learning speed.

To allow for this combination of learning and imitation, a carefully designed architecture is necessary. This thesis deals with this challenge by presenting an architecture designed carefully to support this combination (Chap. 3 – 7) prior to describing the imitation approach (Chap. 8 – 9). In contrast to existing approaches, this thesis does not aim to imitate for the sake of copying another robot's low-level behavior, but to increase the overall learning speed of the robot by imitation. This will have to include all levels of abstraction, not only of low-level behavior. The presented architecture will address this.

The robots using this approach have to infer the answers to the five big questions described earlier by themselves. As already mentioned, the "whom" has so far been ignored using only homogeneous robot groups. This thesis presents an approach that answers this question in the form of a new measure of behavioral difference that can be used prior to the imitation process (Chap. 9).

The "when", meaning the corresponding time frame for imitation, is found out by the externally visible signals that provide information about the overall state of the imitatee. This mimics the expressions of emotion in humans, which guide other human imitator when they want to decide whether to imitate or not.

The "what" and "how", which is the correspondence problem, will be solved by learning actions that are learned against a goal function, which in turn is used to recognize the action itself in the observation data (Chap. 6).

CHAPTER 3

Architecture for learning and imitating in groups

This chapter provides an overview of the architecture that is the basis for realizing self-organizing autonomous systems that are able to learn individual behavior, detect imitation possibilities, and then are able to imitate each other [9, 16, 10, 21]. The architecture enables a robot to individually learn to use its capabilities in dynamic environments. In order to improve the learning speed it has to combine individual learning with imitation of successful behaviors of teammates – with the effect that beneficial knowledge is spread in a robot group. In addition, an imitator shall be able to detect which other robots it should imitate. This is important in heterogeneous robot groups, where the robots have the same goal but different capabilities. This leads to the following requirements for the architecture:

- To enable a robot to learn behavior individually in continuous time and space with noisy sensors and unreliable actuators.
- The ability to adapt and improve the robot's behavior at runtime.
- The maintenance of learned behavior in a form that provides sufficient information and corresponding methods, which are necessary to categorize observed behavior of other robots in order to imitate.
- To support the robot programmer in specifying the overall behavior of an individual robot in an intuitive way.

3.1 Architectural overview

The architecture (Fig. 3.1) is based on Nilsson's Triple Tower architecture [136] with the main components *perception*, *modeling*, and *action* (Fig. 3.1). They define the three basic components for the robot's data flow in each step: First the robot perceives its environment using its sensors, which is done in the perception component. It is responsible for refining the raw perception $\mathcal{I}$ and providing the perception data in the required form to the modeling tower. This component carries out the main reasoning. Based on past experience and current perception, it determines the best next action to achieve its goals. The chosen action is then applied to the actuators using the action component. This chapter introduces the three layers of the model tower that are necessary to individually learn autonomous behavior in continuous time and space with noisy sensors and unreliable actuators. In this sense, behavior is regarded as a complex sequence of reactive low-level skills.

This thesis focuses on the modeling tower. It is composed of three layers with different levels of abstraction that will be presented in more detail in the three following chapters: the *motivation layer* allows the intuitive specification of complex goals (Chap. 4), the *strategy layer* is responsible for devising strategies that fulfill the goals (Chap. 5), and the *skill layer* autonomously learns low-level behavior according to the robot's capabilities that serve the strategy (Chap. 6). These layers interact in such a way that they combine top-down goal specification with bottom-up exploration of the robot's own capabilities.

Each layer in the modeling tower is provided with perception data from the perception tower. The skill layer, however, is the only layer that is directly acting on the environment. Each layer is allowed to request the raw perception $\mathcal{I}$ to be individually preprocessed. Throughout the thesis, the individually preprocessed perception is denoted by $\mathcal{I}_m$ for the motivation, $\mathcal{I}_s$ for the strategy, and $\mathcal{I}_a$ (action) for the skill layer. When defining the three procedures that preprocess the perception, on the one hand the designer must take care to provide enough information for the layers to accomplish their task. On the other hand, he must keep in mind that the learning algorithms can be subject to the curse of dimensionality if too much information is provided. In addition, the preprocessing step must be efficient as it is executed at each processing loop cycle.

At the top level, the motivation layer defines the overall goal in the form of sub-goals. Each sub-goal is defined by one motivation function, which in turn is coupled to one strategy learning algorithm. A motivation function determines, which goal is the most profitable one to reach at each moment. With different motivations, the architecture is able to handle changes in the environment without the need of relearning everything. The middle layer realizes the strategies necessary to accomplish the goals defined by the motivation. It does so on an abstract level, where it views the behavior as symbolic actions to be performed for a certain duration. They are grounded by the skills in the lowest layer. A skill is defined by a goal function and handled by the lowest layer. Using this function, a skill is also capable of recognizing whether a skill similar to itself has been executed in the observations.

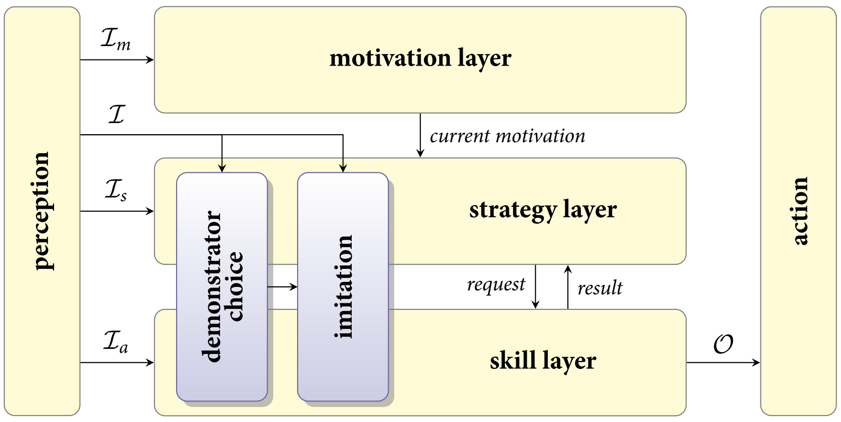


Figure 3.1: The robot architecture

3.1.1 Motivation layer

The overall goal of a robot is specified in the motivation layer by the robot programmer. It can be defined through a number of individual so-called *motivations*, which can even be mutually contradicting.

The fulfillment of a sub-goal is realized by minimizing the corresponding motivation, which is represented by a non-negative scalar. The motivation's value is dependent on external stimulation and the internal reaction to it. Everything that can be physically perceived is regarded as external stimulation. This includes, e. g., the relative position to an object or the robot's battery state. The robot's motivation is defined as a function of the current perception, the time, and the motivation's previous value. Therewith, it is possible, e. g., to specify which perceptual states are preferable or to model impatience so that the robot is preferring behavior that achieves its goals faster. The internal reaction is defined by dynamic evaluation functions that allow the modeling of automatic decay or increase of the motivation. In order to reach desirable states, the system proactively has to select the proper sequence of actions or behaviors in its behavior repertoire that will result in positive evaluations and keeps the motivations low, also called satisfied. If all motivations are satisfied, one can define a special motivation that enforces curious exploration. This can be compared to the children's play instinct. Thereby the robot has a defined motivation at each time step.

3.1.2 Strategy layer

The strategy layer's task is to find state-action sequences that keep the robot's motivations low, which results in achieving its overall goal. It receives a motivation vector containing the individual motivations from the motivation layer. In the strategy layer, complex policies are learned that satisfy the motivation. A policy is a mapping from a system state to an action the robot has to execute. The strategy maintains one policy for each motivation. The strategy layer is moderating between the individual and possibly contradicting goals, which can be ordered and assigned a priority according to how much they're unsatisfied. This means that the robot chooses to follow a policy, which is connected to the motivation with highest priority. This is also the least satisfied one.

For each policy, the strategy layer has to maintain a reasonable state abstraction. That means, that the strategy layer autonomously has to find a state space that is suited for the task at hand. The strategy layer does this by heuristically splitting and merging states.

3.1.3 Skill layer

The policies in the strategy layer treat the actions as abstract symbols without bothering with their actual execution. To have any effect on the environment, they must be grounded physically. This is accomplished in the skill layer. It autonomously learns and maintains a set of skills that achieve their individual goals. A goal is represented by a goal function that measures the achievement of a skill. Each time the skill layer has reliably learned a new skill, it notifies the strategy layer, which in turn updates its own set of abstract actions. For each skill, alternative types of model functions can be provided. They are approximated at runtime and compete by means of their prediction accuracy for being executed at the next time step. Execution in this context means that the chosen approximated function takes the current state as input provided by the perception and returns an action vector.

The action vector, which is sent to the action tower, contains one element for each hardware actor to control. The approach does not consider the low-level hardware part. It just issues the action vector and takes the final realization by means of, e. g., PID controller, etc. for granted.

3.2 Layer interaction

To allow for sufficient reactivity while being able to timely maintain the data structures that are related to the reasoning and involved in the strategy building, the layers work in parallel. The may work even at different frequencies, as shown exemplary in Fig. 3.2. The skill layer runs at higher speed to ensure that the robot reacts appropriately to the environment's events. It recurrently maps the current perception with the skill requested most recently by the strategy layer to the best action. The strategy layer does not necessarily need to run at the same frequency. In most cases it suffices to run the strategy at a much lower frequency. And even then it will not necessarily require a skill change at each time step. In addition, its calculation may take longer once in a while, because it has to reexamine its complete experience history from time to time in order to adjust its data structures. This is being enabled by decoupling the reactive from the reasoning components.

This is exemplary demonstrated in Fig. 3.2 (page 22). The strategy step is triggered to perform its next cycle consisting of determining the current motivation and the corresponding next strategy action ❶. The motivation and strategy layer work synchronously as the strategy layer requires the most current motivation as feedback regarding its last chosen action. The strategy step does not have to finish before the next skill step is triggered ❷. If it is triggered, it simply executes according to the action most recently delivered by the strategy layer. This can be seen in the figure, where "determine next strategy step" in the strategy step is not finished before the skill step is started. In the middle of the skill step ❷, the strategy layer has determined the next action to execute and signals this to the skill layer. Both subsequent skill steps ❸ and ❹ then perform this action accordingly.

3.3 Imitation in robot groups

Individual learning is possible with the three layers alone. In groups of robots with the same overall goal, they should be able to benefit from the results of each other's individual learning processes.

The *imitation* component in Fig. 3.1, which will be described in detail in Chap. 8, allows for imitation in such a group without disturbing other robots. This is usually not the case in typical imitation approaches in literature. There, a known imitatee[1] (human or robot) repeats the same action over and over again. The imitator (robot) then records the multiple action performances and tries to find a generic representation for the underlying behavior.

In contrast to that, the approach in this thesis allows for sporadic imitation. This means that each robot is allowed to observe as much as it wants to, but never to interrupt another robot by requiring a repetition of a behavior it has previously seen. Unlike the traditional imitation approaches, a start and end point of the interesting interval to be imitated is not provided. Therefore, this approach requires that the robots express some kind of overall state. This is equivalent, e. g., to expressed emotions, by which humans are able to infer from each other whether a certain action previously performed was beneficial or not.

A sporadically imitating robot monitors all other observable robots. Once, it has detected a significant change in the expressed overall state of another robot, it analyzes the past observed behavior of that robot. The only assumption made in this process is that the behavior sequence, which has led to a change in the overall state of the observed robot, will also lead to similar changes for the imitator when it executes it. It thereby assumes, that all robots in the group share the same overall goal and have similar motivations. Everything else is left undetermined: The presented approach makes no assumptions about the other robot's strategy implementation, low-level skills, or hardware morphology.

This is only made possible by the three-layered architecture, where the individually learned skills of the skill layer analyze the observations to detect themselves in the observed behavior. The imitation process then tries to find a possible sequence of states in the strategy layer that transfer one to another based on the skills' evaluation. This leads to a condensed *interpretation* of the observation, by means of the imitator's own strategy and skill knowledge. With this state-action-sequence the imitation process then feeds the strategy layer that updates its strategy accordingly. In a way, this is technically similar to the mirror neuron system's way of imitation [150], where each skill tries to recognize its own effects in the observation stream.

[1]Throughout this chapter "imitatee" and "demonstrator" will be used interchangeably.

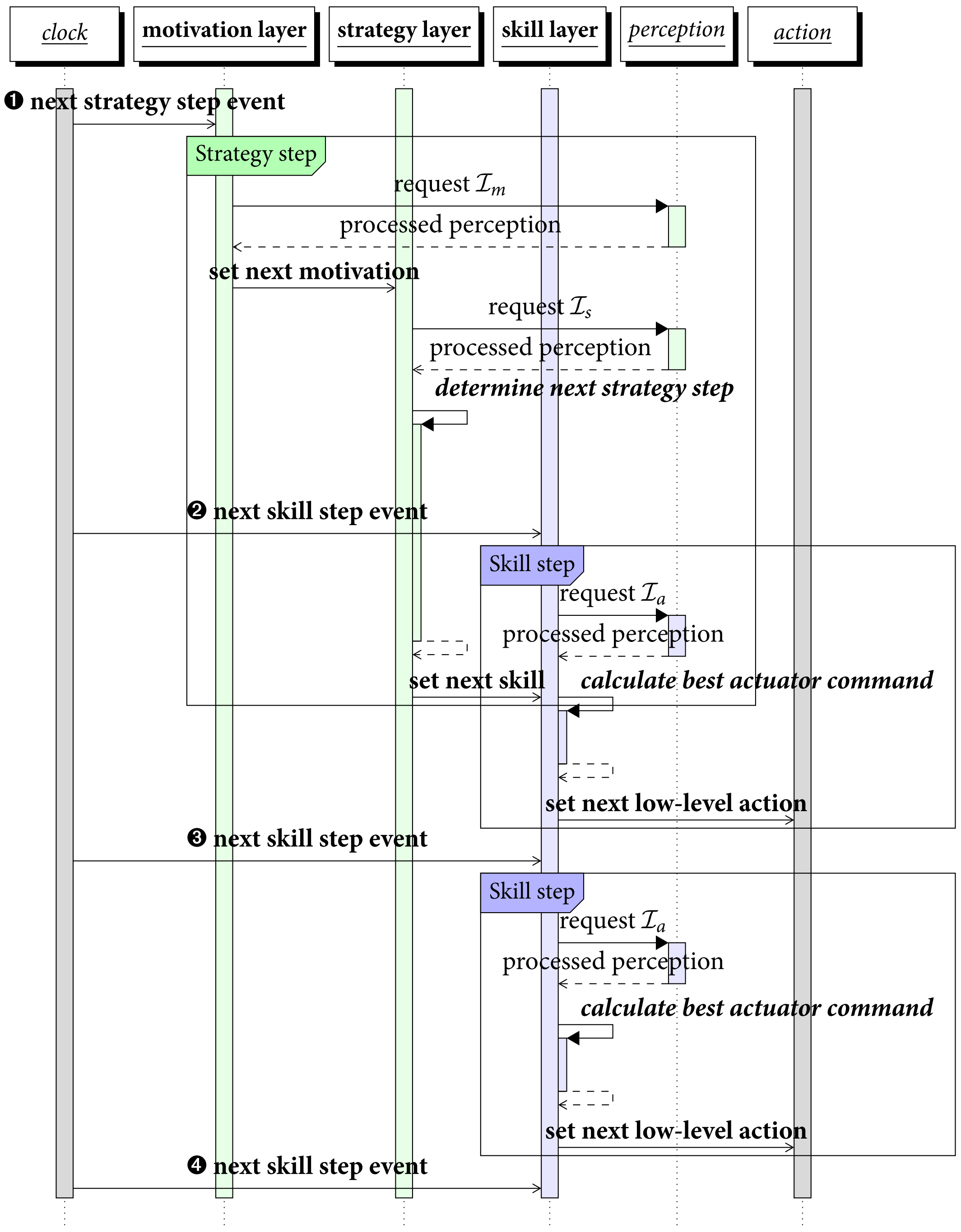


Figure 3.2: Exemplary layer interaction in normal execution mode

3.4 Choice of the imitatee

A major challenge results from sporadic imitation in robot groups: How does a robot know whom it should imitate if no assumptions are made regarding the robot morphology and resulting capabilities? As the observed behavior is fed as an additional experience into its own strategy, imitating arbitrary robots will not lead to long-term performance decrease. By virtue of the layered approach the behavior will be unlearned automatically if it does not result in increased performance, which can be measured directly in the strategy layer. Nevertheless, imitating arbitrary robots might render imitation useless.

Therefore, a robot should only imitate robots with similar capabilities. Observed behavior will then more likely lead to the same outcome. The question is, how to measure the robot similarity. This thesis presents an approach that allows robots to calculate a similarity distance based on affordances, which are interaction possibilities the environmental objects present to a robot (Chap. 9). This is done in the *demonstrator choice* component in Fig. 3.1. As a consequence, the underlying software or hardware specifications of the robots in that group are not important. As long as the affordances are similar, a robot can assume that it will be beneficial if it imitates the other corresponding robot.

As can be seen in the demonstrator choice component in Fig. 3.1, the robot is always monitoring the other robots' capabilities through its perception. Each time the imitation component has detected an interesting and beneficial behavior sequence in another robot's performance, it asks the demonstrator choice component how likely it is that imitating this behavior will lead to a similar performance. Only in the positive case the observed behavior will be imitated.

3.5 Scenarios

As described in the introduction, this thesis' evaluations are oriented towards the prey retrieval scenarios, specifically the Capture-The-Flag variant. There, a number of robots share the same goal of collecting objects in the environment and delivering them to one of several goal bases. Although the robots may be of different size, power, or morphology and thus possess diverse capabilities, they share the same overall goal.

This is supported by the architecture, as it allows the specification of similar goals in the motivation layers of the individual robots', but allows the robots to develop their own strategy and skills – whichever are appropriate to their physical conditions.

In the next three chapters (Chap. 4 to 6), the motivation, strategy and skill layers will be described in detail. Subsequently, the thesis explains how sporadic imitation (Chap. 8) and the choice of the demonstrator (Chap. 9) is realized.

CHAPTER 4

Motivation layer

The motivation layer (Fig. 4.1) defines the goals of the robot. At each instant of time it clearly defines the current needs of robot. With its underlying strategy layer and skill layer it has to choose the behavior that satisfies all goals. By specifying the motivation system, the human designer pinpoints, which behavior the robot has to learn.

Besides the definition of the robot's goals, the motivation layer signals its current overall state to other robots, i. e., its estimation of how well it has proceeded so far in achieving its goals. This will be used by other robots to decide whether they shall imitate this robot or not.

In summary, the motivation layer has to fulfil and support the following tasks:

- Provide information about the robot's current goals and their levels of achievement to the strategy layer.
- Provide a way, by which the robot can be observed by other robots in the group for the purpose of imitation.

In the following, a short background on motivation systems is given. Subsequently, the motivation system of this thesis is described [10, 21].

4.1 Background

The field of motivation and its large body of research have developed over 100 different definitions of the term *motivation* [99]. In this section, a view on motivation is given relevant to robotics.

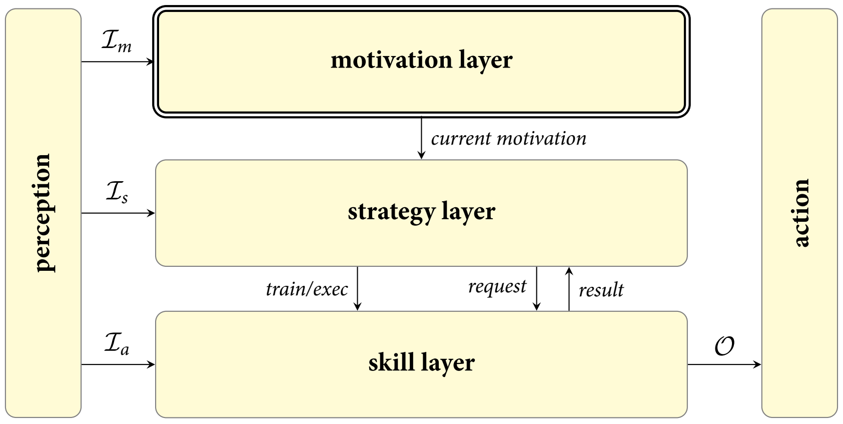


Figure 4.1: The layered robot architecture

4.1.1 Motivation in biological autonomous systems

The actions of autonomous systems, including organisms like humans or animals, have to be guided by basic goals. In nature, these goals are encoded in terms of *drives*. Drives are to be satisfied so that the organism "feels" content. They indicate if something, some parameter of the system itself or the situation within the surrounding environment, is not within normal bounds. The organism then has to take proper action.

One of the first researchers who categorized drives was Maslow. He developed the concept of a *hierarchy of needs* [118] according to which the different drives of a human belong to one of five priorities:

1. **Physiological needs** are the basic needs a human has, such as breathing, sleeping or the need to eating.

2. **Safety needs** address the security of the human's body or of its property and employment.

3. **Social needs** are characterized as the need for friendship and family.

4. **The need for esteem** comprises self-esteem as well as the respect of others.

5. **Self-actualization** is the need with the lowest priority. Examples of self-actualization include abstract concepts like creativity, morality or problem solving.

The four needs with the highest priority are called *deficiency needs*. They definitely must be satisfied before a human may be content. When a deficiency need is satisfied, there is no more incentive to act towards the satisfaction of it. The self-actualization need is called a *growth need*. The growth need can never be fully satisfied, in contrast to the deficiency needs.

Needs are evoked or depleted by *stimuli*, which can be either internal (sleep) or external (food). This means that, e. g., the absence of food over a longer period of time evokes the desire to approach exactly that stimuli. This is achieved by so-called *motivations*.

According to the neuroscientist Salamone [154, 155], a motivation is a set of processes by which "organisms regulate the probability, proximity, and availability of stimuli", including both internal and external stimuli. Salamone distinguishes two phases during the course of motivated behavior: in the *instrumental phase* the "organism regulates the proximity or delivery of stimuli". It is followed by a *terminal phase*, in which the organism directly interacts with the stimulus. Salamone ascribes directional and activational aspects to motivations, as they are typically directed towards a stimulus or away from it and can be of different strength. As Cofer puts it [61]:

> *Motivational concepts, then, have had at least two major functions with respect to behavior. One is to energize responses, either in general or specifically, and to control their vigor and efficiency. The other is to guide behavior to specific ends, i. e., to give direction to behavior.*

Motivations are, therefore, tightly related to reinforcement signals. In the view of the psychologist Spence [169, p. 29]:

> *The combination of a motivating state and the environmental situation impels the subject to respond and to continue responding to various aspects of the situation until a reinforcer is obtained or until removed from the situation.*

Such a *reinforcer* has the capacity to direct behavior, as noted by the biologist Tapp [176]: Stimuli, e. g., to which the organism approaches, are positively reinforcing.

In summary, the need of an organism to satisfy its drives generates its motivations. Those motivations in turn are not diminishing until the corresponding reinforcers are obtained or removed. Transferred to the robotics domain, with the definition of the robot's drives, the robot can be provided with dynamic motivations. These motivations guide the robot towards learning behavior that evokes stimuli, which the robot is seeking and to eliminate those stimuli that are not wanted.

4.1.2 Use of motivation in robots

The neuroscientist Rolls argues that human brains are designed around reward and punishment evaluation systems. In his view, it is the way that genes can build a complex system that will "produce appropriate but flexible behavior to increase fitness" [151]. From the view of the robotic system designer, this insight can be used to circumvent the difficult design of the reward or fitness function, which is typically inevitable in all learning based optimization approaches. Instead of specifying reward functions that guide the learning effort, one can design the response of the motivation system to extrinsic (a ball is visible to which the robot has to drive) and inherent (the battery charge condition of the robot) states. The change in the motivation state is then used to calculate the reward.

Motivations may not only be used to directly control the behavior of autonomous systems like robots. Moreover, they can also be used to control which behavior will be imitated in a group of robots. The benefit of this approach has been shown by Broekens in an experiment where a foraging robot was able to speed up its learning efforts when being guided by human facial expression [47]. In the respective experiment, the recognized human emotions were used by the learning robot as social cues for the desirability of an action.

Therefore, the motivation states of the other robots, which each robot of the group can perceive, will be interpreted internally as a reinforcement signal in this thesis. This is in the sense of Broekens, except that in contrast to his approach, the demonstrating robot does not intend to give reinforcement to another robot. Instead, all robots always output their own current motivation state, which can be used in the imitation process of the imitating robot.

As the robots will output their current motivational state so it can be perceived by nearby robots when they are imitating, they are effectively setting up an *affective communication channel* [140]. This communication channel and the assumption that all robots in the group share the same overall goals are the only assumptions made in this approach.

The behavior learning aspect will be explained in the next two chapters 5 and 6. How drives and thereby motivations can be designed so that they make sense to a robot will be described in the following, using *drive* and *motivation* interchangeably.

4.2 Design of a robotic motivation system

For the evaluation of the robot's overall state the motivation layer uses biologically inspired evaluation methods similar to the motivation described earlier. With them, one is able to specify the overall goal in the motivation layer (Fig. 4.2) as a motivation vector $\boldsymbol{\mu}$:

$$\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)^T \ , \quad \mu_i \in \mathbb{R}^+ \tag{4.1}$$

Each motivation μ_i corresponds to one goal i, which is considered accomplished or satisfied if $0 \leq \mu_i < \mu_i^\theta$, with μ_i^θ defining the threshold of the *well-being region* (Fig. 4.2 and 4.3). The value for μ_i is calculated by the function

$$\hat{\mu}_i : \mathcal{I}_m \to \mathbb{R}^+ \ , \tag{4.2}$$

which is a mapping from the perception at a given time t, $I(t) \in \mathcal{I}_m$, to the degree of accomplishment of goal i:

$$\mu_i = \hat{\mu}_i(I(t)) \tag{4.3}$$

By specifying $\hat{\mu}_i$, which determines the development of μ_i, and its satisfaction threshold μ_i^θ, one is able to intuitively define the robot's overall goal:

$$\boldsymbol{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{pmatrix} = \begin{pmatrix} \hat{\mu}_1(I(t)) \\ \hat{\mu}_2(I(t)) \\ \vdots \\ \hat{\mu}_n(I(t)) \end{pmatrix} \tag{4.4}$$

The robot, then, accomplishes the overall goal comprising all sub-goals, by minimizing each goal i's motivation value μ_i. While it is adapting its strategy and skill set, it does so with only this urge in mind.

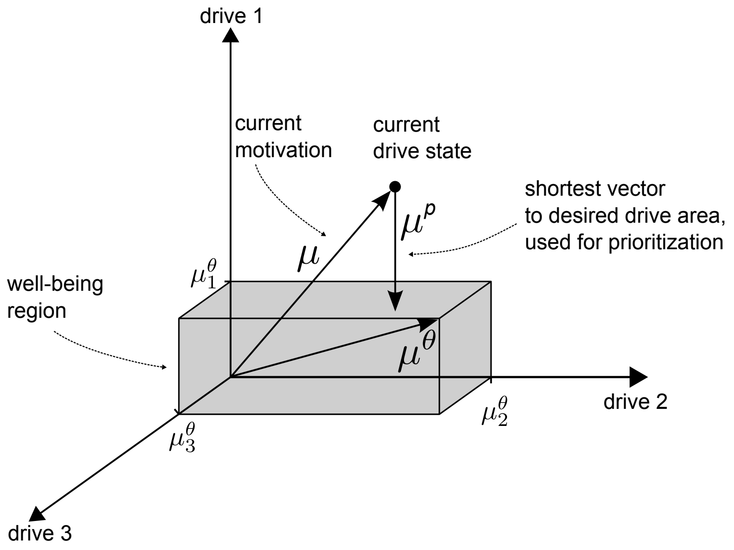


Figure 4.2: An example of a motivation system for three sub-goals: each drive measures the status of accomplishing one sub-goal (0 = fully accomplished). The current motivation $\boldsymbol{\mu}$ is the vector to the current drive state. A drive i is called satisfied, and thereby its goal achieved, if the corresponding motivation element μ_i is below its threshold μ_i^θ. In this example, drives 2 and 3 are satisfied.

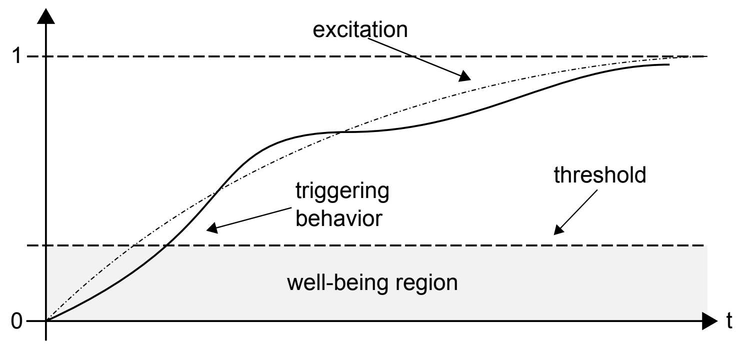


Figure 4.3: An example of a sub-goal subjected to an excitation. The excitation describes the force, which the current drive state is subjected to. By specifying it dependent on the perception and on the internal state of the robot the user is “programming” the final behavior.

When specifying the motivation system of the robotic, one has to ensure that the motivation correctly reflects the achievement status of the robot's goal. That means, that big changes in $\boldsymbol{\mu}$ also reflect big changes in the goal's status of achievement.

4.2.1 Excitation

In addition to the dependence on perception, the motivation can be subjected to time dependent changes. Examples for this are typical human drives that are increasing recurrently, like the need to eat or to sleep.

The motivation layer uses time-dependent excitation for this effect, as it is shown exemplary in Fig. 4.3. Therefore, it is required that the current time is included in the perception $I(t)$. With the excitation, time dynamic behavior can be realized. E. g., an exploration drive could be specified to force the robot avoiding boredom. It would be reinforced each time the robot did nothing, and only decreased if the perception sufficiently changed.

4.2.2 Prioritizing goals

At each time step, the motivation layer provides the current motivation vector to the strategy layer. As will be described in the next chapter, the strategy layer will have to prioritize, which of the sub-goals are to be handled first. This is done based on the shortest vector $\boldsymbol{\mu}^p$ of the current motivation state to the well-being region, which is used for drive prioritization (Fig. 4.2):

$$\boldsymbol{\mu}^p = \begin{pmatrix} \max(0, \mu_1 - \mu_1^\theta) \\ \max(0, \mu_2 - \mu_2^\theta) \\ \vdots \\ \max(0, \mu_n - \mu_n^\theta) \end{pmatrix} \tag{4.5}$$

In order to model a hierarchy of needs, the different drives can be prioritized by means of an according scaling.

4.3 Conclusion

The robot interprets the minimization of the vector of the point of origin to the current drive state as its current motivation. This serves two functions in the framework as required at the beginning of this chapter:

1. On the one hand, the motivation $\boldsymbol{\mu}$ is used by the strategy layer to calculate the reward. Thereby, a positive reinforcement is given to the strategy layer, whenever the motivational state approaches the zero vector and thus moves towards the well-being region. A negative reinforcement is given in the opposite case.

2. On the other hand, it supports imitation in multi-robot scenarios. The motivation value in this motivation layer can be used to express the robot's overall well-being to the other robots and guides them when they are observing each other to imitate only obviously beneficial behavior.

In summary, the following concepts are realized by the motivation layer:

- The overall goal of a robot is split into multiple sub-goals and specified by means of drives.
- A drive represents a need, which the robot wants to satisfy.
- A drive has a threshold, which marks whether the drive is satisfied. The strength of the desire to satisfy the need is represented by the motivation.
- All robots in a group are assumed to have a similar set of drives, meaning that they share the same overall goal.

An example for a concrete realization of the motivation layer will follow in Chap. 7, after the strategy layer (Chap. 5) and skill layer (Chap. 6) have been introduced.

CHAPTER 5

Strategy layer

In order to satisfy the motivation layer with its motivation vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_n)^T$ and the well-being region defined by $\boldsymbol{\mu}^\theta$, the robot has to derive a strategy that is able to keep $\mu_i < \mu_i^\theta$ given only the experience stream

$$\langle \dots, (o, a, d, \boldsymbol{\mu}, f)_{t-1}, (o, a, d, \boldsymbol{\mu}, f)_t, \dots \rangle \tag{5.1}$$

with the following notation:

observation o**:** This is the preprocessed and filtered raw perception $\mathcal{I}$ representing the fully observable state ($o = \mathcal{I}_s$)[1],

action a is the action that is sent to the skill layer in order to execute the corresponding low-level behavior.

duration d is the duration between two consecutively triggered actions. The duration is used to properly discount the received reward, which is delivered in terms of the motivation vector $\boldsymbol{\mu}$.

motivation $\boldsymbol{\mu}$**:** The time derivative of the motivation vector is used to calculate the return of the last action.

failure f signals whether the skill layer (Chap. 6) considers the last executed action as failed. This can be, for instance, the skill layer signalling that the previously executed skill has not performed as expected, because the robot is trying to drive against a wall.

[1]In domains, where this cannot be assured, the presented approach still works, but the underlying SMDP framework has to be extended to a *partially observable Markov decision process* (POMDP) [165]. For more details, cf. Sec. 5.1.1.

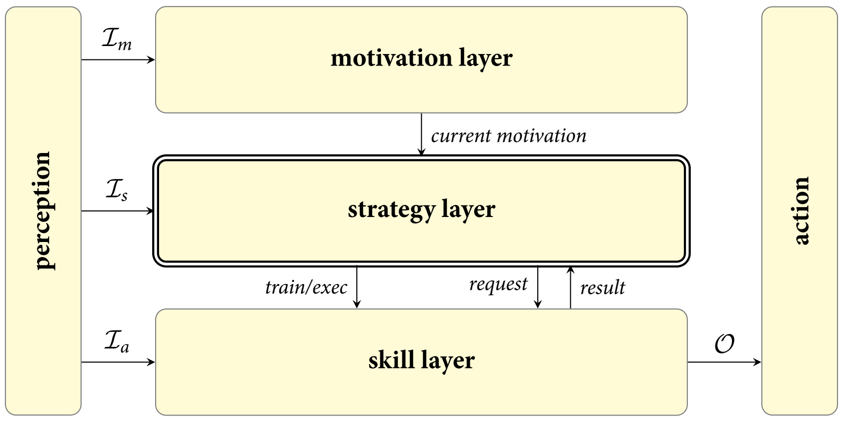


Figure 5.1: The layered robot architecture

The derivation of a strategy is the task of the strategy layer [21, 10], which is located between the motivation and the skill layer, as shown in Fig. 5.1.

To keep this learning program feasible, the strategy layer is not trying to learn one strategy for the whole motivation system. Instead, it is generating one strategy for each motivation μ_i. The system then selects the active strategy dependent on the dynamic drive prioritization (Eq. (4.5) in Sec. 4.2.2). A simple approach is to choose the drive with the least satisfied motivation. This section will restrict the description to the strategy of one sub-goal i and the corresponding motivation $\mu = \mu_i$.

The strategy learning approach is outlined in Fig. 5.2 and works as follows. After preprocessing the raw perception and appending it to its accumulated experience, the strategy generalizes the raw state, which is the actual state observations o_t, into an abstract region s_t. This is vital, because operating directly on the raw state space is infeasible. The solution space would be too big to be explored within a reasonable amount of time. The abstraction is achieved by a set of heuristics that modify the mapping, which assigns abstract states to the preprocessed perception states. The mapping may be of any form of abstraction method. In this thesis, the approach uses nearest neighbor, as it is one of the most general abstraction methods [62]. Each time new experience is made by the robot, the model consisting of the state transitions, rewards, and time statistics is updated. Using the abstract states, the approach then has to find a sequence of state-action-pairs that leads the robot to its current goal. This problem can be cast into the Markov decision process (MDP) class. Therefore, reinforcement learning is used to find the optimal strategy according to the imperfect perception experience [172]. In summary, a model-based reinforcement learning with prioritized sweeping [128] is used to derive an optimal policy by means of semi-Markov decision processes (SMDP) [147] (cf. Sec. 5.1.2). SMDPs are necessary, as they allow – in contrast to MDPs – for variable-duration actions, which is necessary for realistic scenarios. The policy can be queried at each time step for the best action according to the current abstract state, which can be efficiently realized by a simple look up table. From time to time, random actions are used instead to ensure a continuous exploration of the strategy space.

The strategy layer does not execute actions by itself. Separating the concerns of high-level strategy and low-level actions, the strategy layer only treats them as symbols and sends these to the

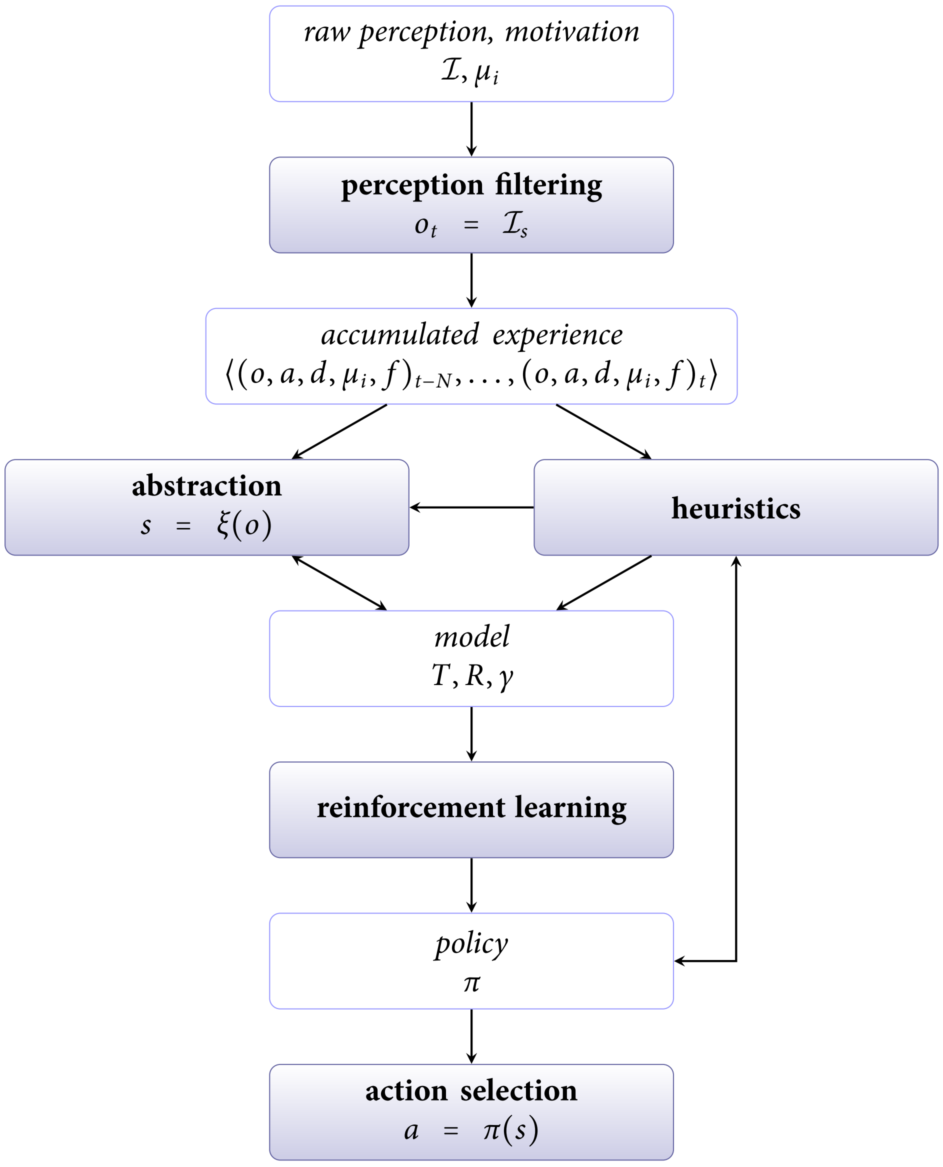


Figure 5.2: Processes involved in updating one policy in the strategy layer

skill layer, which translates them into the particular actuator settings and executes them in the environment.

After the presentation of reinforcement learning basics and the description of the state of the art in this domain, this chapter presents the strategy learning component in detail. As most of the notation can be best explained within the context of reinforcement learning, the chapter will begin with a description thereof (Sec. 5.3). The chapter continues by explaining how the state is abstracted (Sec. 5.4) and how it is creating and maintaining models using the abstracted state space (Sec. 5.5). The remainder of the chapter addresses practical issues and concludes with an example.

5.1 Background

The problem of finding an optimal action sequence that reaches a goal in an unknown environment is known as *sequential decision making under uncertainty* [147]. In the case that a robot is placed in such a problem setting and no supervisor or guide is available, all information the robot can rely on is the perception $s \in \mathbb{S}$, the executed action $a \in \mathbb{A}$ and some kind of "outcome" of it, called reward $r \in \mathbb{R}$. The robot's experience stream thus looks like the following:

$$\dots, (s_{t-2}, a_{t-2}, r_{t-2}),\ (s_{t-1}, a_{t-1}, r_{t-1}),\ (s_t, a_t, r_t) \tag{5.2}$$

If each state of the state space conveys all information necessary for the robot to intelligently pursue its goal, the state space $\mathbb{S}$ is said to be Markovian, i. e., it has the Markov property. In this case, the underlying structure of the problem is a Markov process. The problem can then be modeled as a Markov decision process (MDP) and solved with reinforcement learning approaches [172][2]. The following sections present the MDP as the most basic case and introduce the semi-Markov decision process (SMDP) as a more general MDP, which is suitable for time-dependent applications like the robot scenarios in this thesis.

5.1.1 Markov decision processes

A *Markov decision process* (MDP) [147] is defined by the tuple $(\mathbb{S}, \mathbb{A}, T, R)$ and describes a control problem where an agent interacts with its environment in order to optimize the reward it receives from it:

- $\mathbb{S}$ is the robot's finite state space.
- $\mathbb{A}$ is the finite set of actions the robot can execute.

[2] In non-Markovian scenarios, *partially observable MDPs* (POMDP) have to be used [165]. Although POMDPs are non-Markovian, the optimal POMDP solution is Markovian over the belief state, which is an approximation of the underlying hidden state.

- $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}+$ is the transition function. $T(s, a, s')$ defines the probability that action a executed in state s leads to the next state s', where

$$\forall s \in \mathbb{S}, a \in \mathbb{A} : \quad \sum_{s' \in \mathbb{S}} T(s, a, s') = 1 \bigvee \sum_{s' \in \mathbb{S}} T(s, a, s') = 0 \,.$$

- $R : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the reward function, which provides feedback about the outcome of the robot's last action in its last state.

In real-world applications T and R cannot be given to the robot directly, but have to be found out by the robot via interaction. The decision process is called *Markov*, because both functions depend only on the current state and action, and not on their past history. I. e., the probability $P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t)$ of transitioning to state s' and receiving reward r after the robot has executed action a_t in state s_t is independent of the robot's history:

$$P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t) = P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \dots, r_1, s_0, a_0) \tag{5.3}$$

5.1.1.1 Policy

A *policy* defines which action the robot chooses in a given state. This is the actual strategy of a robot. It is a mapping $\pi : \mathbb{S} \to \mathbb{A}$ that assigns an action $a \in \mathbb{A}$ to each state $s \in \mathbb{S}$. The goal of reinforcement learning is to find an optimal policy in the policy space Π for the Markov decision process.

Optimality is defined in terms of the *value function* $V^\pi : \mathbb{S} \to \mathbb{R}$ that estimates how useful it is for a robot to be in a given state, also known as the *utility*. A policy is called optimal and denoted by π^* if the following condition holds for all policies $\pi' \in \Pi$: $V^{\pi^*}(s) \geq V^{\pi'}(s) \ \forall \ s \in \mathbb{S}$. When defining the value function, the incorporation of the reward has to be tailored to the class of application domains, as this defines the optimization criterion. Most often used are the *average*, *finite horizon*, and *discounted* reward [173].

average Some applications ask for the maximization of the long-term average reward. With $E[\cdot]$ being the expectation, this is realized by

$$V^\pi(s) = \lim_{N\to\infty} \frac{1}{N} E\left[\sum_{t=0}^{N-1} r_t \mid s_0 = s\right] \tag{5.4}$$

finite horizon If the robot's strategy horizon is finite with a fixed life time N, i. e., where a continuous task has to be performed over a given time, the reward can simply by accumulated:

$$V^\pi(s) = E\left[\sum_{t=0}^{N-1} r_t \mid s_0 = s\right] \tag{5.5}$$

discounted For continuous but not time-limited tasks it is wise to give a smaller weight to a reward that is more distant in the future. This can be achieved with the *discount factor* $\gamma \in [0, 1)$:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right] \tag{5.6}$$

As this thesis is concerned with learning and improving the performance of robots in a robot group that have to accomplish continuous tasks that are not time-limited, the following focuses on value functions that incorporate the *discounted reward*.

Although value functions allow to discriminate between more and less useful states, they don't allow yet for decision making. Therefore, it is reasonable to analyze it on the state-action-level, which is done by the *Q value function* $Q^\pi : \mathbb{S} \times \mathbb{A} \rightarrow \mathbb{R}$. It determines the value of executing an action according to a policy π in a given state by considering the received reward and the expected value of the next state:

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') V^\pi(s') \tag{5.7}$$

Once the optimal value function $V^{\pi^*}(s,a) = E\left[Q^{\pi^*}(s,a)\right]$ and with it the optimal Q value function Q^{π^*} is found, the optimal action is retrieved by

$$\pi^*(s) = \arg\max_{a \in \mathbb{A}} Q^{\pi^*}(s,a) \ . \tag{5.8}$$

5.1.1.2 Solving Markov decision processes

The question is how to find V^{π^*} in order to get the optimal policy π^*. This can be done by *dynamic programming* (DP) methods [39, 91]. The most well-known DP methods are *value iteration* and *policy iteration*, which compute the optimal policy for a given model. This model has to be provided beforehand or explored by the robot itself. For this reason, DP methods are called *model-based* or *indirect* in contrast to *model-free* methods that derive the policy directly without building a model.

Value iteration starts with an arbitrary value function V and updates it according to the so-called Bellmann's Equation for all states $s \in \mathbb{S}$

$$V^\pi_{n+1}(s) = \max_{a \in \mathbb{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') V^\pi_n(s') \right] \tag{5.9}$$

until no significant change is detected any more, according to a defined precision ε: $||V^\pi_{n+1} - V^\pi_n|| < \varepsilon$.

Policy iteration starts with an arbitrary policy π, which is used to calculate the value function. With the updated value function the policy is updated:

$$\pi_{n+1}(s) = \arg\max_{a \in \mathbb{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathbb{S}} T(s,a,s') V^\pi_n(s') \right] \tag{5.10}$$

This is repeated until the policy has converged: $\pi_{n+1} = \pi_n$.

5.1.2 Semi-Markov decision processes

In the MDP framework, actions always have unit duration. Processes get more complicated if they are allowed to take variable amounts of time before transferring to the next state. This is inevitable in real-world applications like the evaluation scenarios in this thesis. In that case the Markov property does not hold any more: the current state alone does not suffice to predict the next state. Problems with this characteristic have to be solved in the *semi-Markov decision process* (SMDP) framework [147], which models continuous-time discrete-event systems.

At first this involves replacing the step discount factor γ by a continuous discount factor $\beta \in (0, \infty)$. A reward r received after time t thus leads to a net reward of $e^{-\beta t}$. If $\beta = \infty$ the robot is said to be *myopic*, as the future reward is discounted by $e^{-\infty t} \approx 0$ and the robot thus is concentrating only on the immediate reward. In an MDP world this would be realized by $\gamma = 0$. With β approaching zero the robot is paying increasingly more attention to reward that is farther in the future. In addition, it requires more complex transition models. The transition function T becomes a probability density functions over time in the SMDP framework, in contrast to a simple probability distribution in the MDP context.

5.2 State of the art

Even when ignoring the need for action recognition, which is necessary for perception-based imitations, there is a lot of research done in the area of strategy learning in continuous state and action spaces. It can be divided into *model-free* and *model-based* approaches. Model-free approaches learn the optimal strategy and actions directly from the interaction with the environment. Model-based approaches firstly learn a model of the environment and themselves.

5.2.1 Model-free approaches

It is obvious that a full search in continuous state and action spaces is infeasible. For reinforcement learning approaches to be applied in realistic domains, it is therefore vital to limit the search to small areas in the search space. One approach to do that is the *actor critic* method [105]. It separates the presentation of the policy from the value function. For each state, the actor maintains a probability distribution over the action space. The critic is responsible for providing the reward from the actions taken by the actor, which in turns modifies its policy. As this relieves the designer from assumptions about the value function, it introduces new assumptions about the underlying probability distribution. To overcome this problem Lazaric et al. devised *Sequential Monte Carlo Learning* [109], which combines the actor critic method with a nonparametric representation of the actions. After initially being drawn from a prior distribution, they are resampled dependent on the utility values learned by the critic.

Hasselt and Wiering devised the *Continuous Actor Critic Learning Automaton* approach. It allows robots to use reinforcement learning for operating on continuous state and action spaces [181]. They calculate real valued actions by interpolating the available discrete actions based on their utility values. Therefore, the performance is highly dependent on initial assumptions about the value function.

Bonarini et al. developed *Learning Entities Adaptive Partitioning* (LEAP) [45], a model-free learning algorithm that uses overlapping state space partitions, which are dynamically modified to learn near-optimal policies with a small number of parameters. Whenever it detects incoherence between the current action values and the rewards from the environment, it modifies those partitions. In addition, it is able to prune over-refined partitions. Thereby, it creates a multi-resolution state representation specialized only in areas where the finer resolution is actually needed. The action space is not considered by this approach. In their grid world experiment, they use a fixed set of predefined actions.

5.2.2 Model-based approaches

The *Adaptive Modelling and Planning System* (AMPS) by Kochenderfer [100] maintains an adaptive representation of both the state and the action space. In his approach, the abstraction of the state and action space is combined with policy learning: states are grouped into abstract regions, which have the common property that perception-action-traces, previously performed in that region, "feel similar" in terms of the failure rates, duration, and expected reward. It does so by splitting and merging abstract states at runtime. AMPS not only dynamically abstracts the state space into regions, but also the action space into action regions. This is, however, done in a very artificial way that could not yet been shown to work in real world domains.

Although the strategy layer of this thesis is inspired by AMPS, it differs from it in the following important points: AMPS applies the splitting and merging also to the action space, which works fine in artificial domains but will not cope with the domain dependency one is typically faced with in real environments. In contrast to that, this thesis' approach uses goal functions as the strategy's actions, which have to be realized by a separate skill-learning layer. This leads to a useful separation of concerns: the task of the strategy layer is to find sequences of actions and treats actions as mere symbols. The skill layer by means of data driven skill functions then grounds these symbols.

Another aspect is the supported number of goals. Take, e. g., a system, which has to fulfill a specific task while paying attention to its diminishing resources. While, on the one hand, accomplishing the task, the resources might get exhausted. If it, on the other hand, always stays near the fuel station, the task will not be accomplished. Approaches like AMPS, which do not support multiple goals by multiple separate strategies, have to incorporate all different goal aspects in one reward function. This leads to a combinatorial explosion in the state space and implicates a much slower learning convergence.

As already described, this approach uses abstract motivations, which the designer has to specify. These motivations may also contain competing goals. The major advantage of this approach is that the robot can learn one separate strategy for each motivation. Depending on the strength of each motivation, it has now a means to choose the right strategy for the current perception and motivation state.

5.2.3 Discussion

All these approaches have the following underlying restricting assumptions. First, they assume that optimal actions are either possible to be predefined or effectively learnable within the reinforcement learning framework. This means that prior to using these approaches a careful analysis of all occurring events in the environment has to be carried out by the designer. Except for AMPS, they are all based on Markov Decision Processes (MDP).

Time varying actions, which are the norm in realistic scenarios, however, require a semi-Markov Decision Process (SMDP), which complicates the search in continuous action spaces. Arguing that models are difficult to approximate at runtime, the model-free approaches do not learn a model on which the policy is approximated, but only the value function. Furthermore, they always solve only one goal and it is not intuitively clear, how multiple possibly contradicting goals could be integrated using the same state and action space for all goals. The biggest problem of all, however, is that these approaches are solely aimed at learning from scratch. It is not clear how those could be combined with imitation. The architecture presented in this thesis was designed with these aspects in mind.

The approach in this thesis has the following advantages over AMPS:

- The actions are learned in a developmental fashion, allowing the robot to actively explore its own capabilities. By separating strategy learning from action learning, this approach allows for the application of learning algorithms natural to the respective level of abstraction. This is vital for imitation as it allows for associating observations to belong either to the strategy or skill level, when the robot tries to imitate another robot.
- The model adaptations automatically tune most of their parameters. In contrast to that, AMPS requires the robot to be run in the target environment for several times until the correct thresholds for the different heuristics have been found.
- No "oracle" is needed. AMPS requires to guide the robot to the goal several times until it is able to learn by itself. In this thesis, the motivation system allows the designer to specify many simple reward functions, instead of a complex one. This eases the strategy generation, as many simpler strategies can be learned instead of one complex strategy.

5.3 Policy

The reward in this thesis is composed of two reward elements. The transition reward $r \in \mathbb{R}$ specifies the one time reward for transferring the robot from the abstract state $s \in \mathbb{S}$ with action $a \in \mathbb{A}$ to the abstract state $s' \in \mathbb{S}$. The reward rate $\rho \in \mathbb{R}$ is given continuously for staying in state s while executing action a until the robot arrives at state s. This is necessary to provide the most general form of goal specification by means of the motivation system. Both components can be extracted from the motivation μ_i of goal i by means of

$$(r, \rho) = \begin{cases} (-\dot{\mu}, 0), & \text{if } |\dot{\mu}| > \theta_R \\ (0, -\dot{\mu}), & \text{otherwise} \end{cases} . \tag{5.11}$$

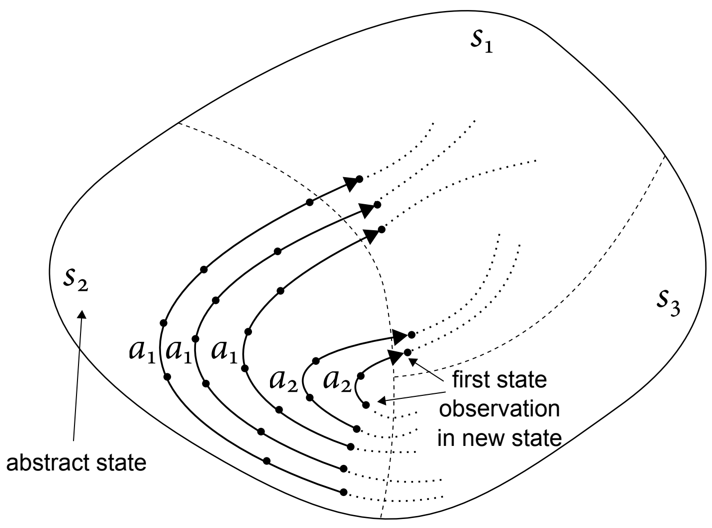


Figure 5.3: This example shows five different paths of different duration from s_2 to s_1 using the actions a_1 and a_2. The reward received, *when arriving* at s_1 is discounted by $\gamma(s = s_2, a_{1,2}, s' = s_1)$. The reward received *while* staying in s_2 and executing action $a_{1,2}$ is discounted by $\lambda(s = s_2, a_{1,2}, s' = s_1)$. Both discounting functions incorporate the time t it takes to transfer from one state to another and its probability $P_t(t \mid s = s_2, a_{1,2}, s' = s_1)$.

This means that the reward is interpreted as transition reward, if it exceeds the reward rate threshold θ_R, otherwise it is received as reward rate. In the following, the notation of Kochenderfer is followed regarding the learning of strategies on abstract state spaces with SMDPs [102].

The strategy layer basically has to account for both types of reward in its discount calculation. This is necessary to discriminate between different actions that result in the same outcome in terms of the resulting state and reward, but involve different amounts of time. This shall be demonstrated with Fig. 5.3 using two different scenarios. Obviously, the robot should prefer faster actions if the result of the involved transition to a different state with positive reward is encountered earlier and can thus be made more often. Let actions a_1 and a_2 in Fig. 5.3 correspond to driving to a goal represented by s_1 in two different ways. In this case, the robot should prefer a_2 over a_1. If, on the other hand, two actions in the same state yield a constant positive reward, the robot should prefer the one that stays longer in the specific state, as this would result in more accumulated positive reward. Let a_1 in Fig. 5.3 now correspond to the action of charging the batteries. Then the robot should obviously prefer that action over some other action a_2 that would leave the state very quickly. Both effects are achieved by the *discounted value of the unit transition reward*[3] $\gamma(s, a, s')$ and the *average cumulative discounted sum of unit reward rates* $\lambda(s, a, s')$, respectively. They approximate the discount factor by which the received reward is multiplied to retrieve the reward that is incorporated into the policy.

Determining the true values for $\gamma(s, a, s')$ and $\lambda(s, a, s')$ of a transition entails the averaging of the discount term $e^{-\beta t}$ (Sec. 5.1.2) over all time durations t that will ever be realized by a robot.

[3]Not to be confused with the discount factor γ in MDP problems.

For $\gamma(s, a, s')$ that is

$$\gamma(s, a, s') = \int_0^\infty e^{-\beta t} \mathrm{d}P_t(t \mid s, a, s') \,, \tag{5.12}$$

where the integral is the Lebesgue integral [88]. $P_t(t \mid s, a, s')$ is the probability that the robot is able to transition from s to s' using action a within time t. It is used to weigh the term $e^{-\beta t}$. $\gamma(s, a, s')$, thereby, calculates the discount factor for the transition process.

As explained above, some situations require to criticize the process of carrying out an action *while staying in the same state*. For this, the *average cumulative discounted sum of unit reward rates* is needed. It is received continuously while executing action a in state s until arriving at state s' and is calculated as

$$\lambda(s, a, s') = \int_0^\infty \int_0^t e^{-\beta t'} \mathrm{d}t' \mathrm{d}P_t(t \mid s, a, s') \,. \tag{5.13}$$

$\lambda(s, a, s')$ differs from $\gamma(s, a, s')$ only in that it involves the integral of the discounted reward rates instead of the simple discounted transition reward.

The expected discounted reward when started in state s and acting according to policy π can be specified as the expectation of the sum of the discounted transition reward and reward rate [101]:

$$V_\pi(s) \equiv E\left\{ \sum_{k=1}^{\infty} \left[\underbrace{e^{-\beta t_{k+1}} r_k}_{\substack{\text{discounted}\\\text{transition reward}}} + \underbrace{\int_{t_k}^{t_{k+1}} e^{-\beta t} \rho_k \mathrm{d}t}_{\substack{\text{discounted}\\\text{reward rate}}} \right] \middle| \, s_1 = s, a_k = \pi(s_k) \right\} . \tag{5.14}$$

The robot has to determine $V_\pi(s)$ iteratively at runtime. It does so by updating recurrently the value function each time a new event occurs:

$$V_\pi(s) \leftarrow \max_{a \in \mathbb{A}} \left[R(s, a) + \sum_{s' \in \mathbb{S}} P(s'|s, a) \gamma(s, a, s') V_\pi(s') \right] . \tag{5.15}$$

The inner term is the compound reward $R(s, a)$ that can be expected in state s when executing action a added to the expected discounted value of the next state that the robot will transition to. $V_\pi(s)$ is then the maximum of the values for all the possible actions available in s.

With the updated value function $V_\pi(s)$, $\pi(s)$ can be assigned the action that yielded the maximum for $V_\pi(s)$:

$$\pi(s) = \arg\max_{a \in \mathbb{A}} \left[R(s, a) + \sum_{s' \in \mathbb{S}} P(s'|s, a) \gamma(s, a, s') V_\pi(s') \right] \tag{5.16}$$

Eventually, $V(s)$ and $\pi(s)$ will then converge to the true value function $V^*(s)$ and optimal policy $\pi^*(s)$, respectively [100].

$R(s, a)$ is determined according to Kochenderfer by the expected sum over all possible states of the transition reward $r(s, a, s')$ and reward rate $\rho(s, a, s')$ discounted by $\gamma(s, a, s')$ and $\lambda(s, a, s')$, respectively:

$$R(s, a) = \sum_{s' \in \mathbb{S}} P(s' \mid s, a) \left(\gamma(s, a, s') r(s, a, s') + \lambda(s, a, s') \rho(s, a, s') \right) \tag{5.17}$$

This can be done with non-parametric estimation [100]. For that it is first necessary to estimate $\gamma(s,a,s')$, as $\lambda(s,a,s')$ is being simplified to

$$\lambda(s,a,s') = (1-\gamma(s,a,s'))/\beta\ . \tag{5.18}$$

If $n(s_k,a_k,s_{k+1})$ is the number of (s_k,a_k,s_{k+1}) transitions, then γ is estimated after the k^{th} transition by $\hat{\gamma}$ as follows:

$$\hat{\gamma}(s_k,a_k,s_{k+1}) \leftarrow \hat{\gamma}(s_k,a_k,s_{k+1}) + \frac{e^{-\beta t_k} - \hat{\gamma}(s_k,a_k,s_{k+1})}{n(s_k,a_k,s_{k+1})} \tag{5.19}$$

Let $\sigma_r(s,a,s')$ be the accumulated transition reward and $\sigma_\rho(s,a,s')$ the sum of the reward rates received when going from s to s' with action a. Using the simplification in Eq. (5.18) and the approximation $\hat{\gamma}(s_k,a_k,s_{k+1})$ in Eq. (5.19), the estimated expected reward $\hat{R}(s,a)$ for executing a in s can then be calculated as [101]

$$\hat{R}(s,a) = \frac{1}{n(s,a)}\sum_{s'\in\mathbb{S}}\left(\hat{\gamma}(s,a,s')\left(\sigma_r(s,a,s') - \frac{\sigma_\rho(s,a,s')}{\beta}\right) + \frac{\sigma_\rho(s,a,s')}{\beta}\right)\ . \tag{5.20}$$

5.4 State abstraction

The state is determined by the state abstraction (Fig. 5.2, p. 35)

$$\xi : \mathcal{I}_s \to \mathbb{S}\ . \tag{5.21}$$

It maps the raw state observations $o_t \in \mathcal{I}_s = \mathbb{R}^d$ in the perception space to states in the abstracted region space $\mathbb{S}$, where d is the number of dimensions of the perception space. This is necessary in order to achieve a feasible number of meaningful states. The strategy uses the raw observations in the experience only to maintain the abstract state space. The state space of the strategy consists of the abstract states maintained by ξ.

At each interaction with the environment, the robot receives a new observation, which it has to assign to one of the states in $\mathbb{S}$. This is a case of instance-based learning and can be solved by nearest neighbor generalization (NN) [62]. Based on a distance measure D, a new instance of data is labelled with the label of its nearest neighbor in the special case of 1NN. This can be generalized to kNN, where the label is determined from the majority vote of the k nearest neighbors'. In that case the votes are typically inversely weighted by the relative distance to the new data point. The nearest-neighbor approach requires the function D, by which the distance between two points is measured, to be a pseudo-metric.

Let $\mathbb{O}$ be the set of observations that are to be mapped and queried afterwards and D a distance function. The tuple $(\mathbb{O}, D)$ is called metric space if D fulfills the conditions of a metric:

Definition 5.1 (Metric) *A function $D : \mathbb{O}\times\mathbb{O} \to \mathbb{R}$ is called metric if it satisfies the following conditions:*

- *Positiveness:* $\forall x, y \in \mathbb{O},\ D(x,y) \geq 0$.
- *Symmetry:* $\forall x, y \in \mathbb{O},\ D(x,y) = D(y,x)$.
- *Reflexivity:* $\forall x \in \mathbb{O},\ D(x,x) = 0$.
- *Strict positiveness:* $\forall x, y \in \mathbb{O}, x \neq y \ \Rightarrow D(x,y) > 0$.
- *Triangular inequality:* $\forall x, y, z \in \mathbb{O},\ D(x,z) \leq D(x,y) + D(y,z)$.

If the distance measure does not satisfy the strict positiveness *condition, it is called* pseudo-metric. *In this case, different observations having a distance of zero will be regarded as the same observation.*

As a robot is observing the state as a real-valued vector, the observations span a vector space, which is a metric space.

The implementation of ξ has to be efficient to be used at runtime. As each strategy step the abstraction will be queried at least once to determine the state for the current observation. This will then be added to the instances used by ξ. In addition, observations might be dropped if they are too old. From time to time the mapping will have to be modified either by splitting or merging the existing states. This requires the implementation of ξ to support the following operations in an efficient manner:

- **Insertion** of new state observations
- **Deletion** of old state observations
- **Querying** most similar state observation to a new state observation

This is possible with so-called kd-trees [42, 74, 75]. A kd-tree is a data structure supporting fast searches in k-dimensional data sets. It accelerates the query speed by leveraging the spatial properties of the data. As it originally is a static data structure, new insertion or deletion of data will degenerate the kd-tree once it is constructed based on the available data. This leads to suboptimal performance. Kd-trees are therefore extended by incremental approaches, which construct balanced kd-trees. The BKD-tree [146] and the hB-tree [115] are two examples. While the hB-tree can suffer from degenerating space utilization [72], the BKD-tree maintains close to 100% space utilization while insertion, deletion, and performing range queries are guaranteed to be of amortized logarithmic time. Basically, it maintains a sequence of balanced kd-trees of increasing complexity.

The general drawback of kd-trees in suffering from the curse of dimensionality, can be alleviated when the search for the nearest neighbor is done with the *Best-Bin-First* approximation algorithm [37]. It finds the nearest neighbour for a large fraction of the queries, and a very close neighbour in the remaining cases. Thereby, it is able to efficiently find the nearest neighbor even in high dimensional spaces.

Because the state observation is domain dependent, the vector space assumption might be violated. In this case, the mapping can be realized by an M-Tree approach, which generalizes over spatial trees by not relying on a strict order defined over all objects: The Symmetric M-tree [160] by Sexton and Swinbank supports efficient dynamic insertion and deletion while only paying attention to the relative object distances and not their order. However, as general metric approaches are targeted towards applications where the distance calculation is the most costly action, they usually cache the distances. This naturally leads to a growing amount of data to be stored. Those approaches are therefore more suited to multimedia applications. One example is the search for the most similar image in an image database. There, the average color of an image could be one feature to be considered in the distance calculation, which is very time intensive.

5.5 Model

At the beginning, all states belong to only one region, since the robot has no reason to believe otherwise. While interacting with the environment, the model is modified by adapting the state abstraction through *splitting* or *merging* states:

split$(s, l_1, \dots, l_n)$ splits the state s into n new states $\{s_1, \dots, s_n\}$. l_i is a list of observations that shall be assigned to the same new state s_i. All remaining observations

$$\{o_k \mid \xi(o_k) = s \wedge o_k \notin \bigcup l_i\} \tag{5.22}$$

are then mapped by ξ to their nearest neighbor of the new states s_i. The original state s is removed afterwards.

merge$(l_1, \dots, l_n)$ takes n lists of observations that are mapped to n different states $\{s_1, \dots, s_n\}$, creates a new state s and maps all observations in $\bigcup l_i$ to that new state. The corresponding original states $\{s_1, \dots, s_n\}$ are removed afterwards.

The following heuristics use *merge* and *split* to adapt the state abstraction ξ and the model including the underlying statistics so that they reflect the world experience.

5.5.1 Transition heuristic

As mentioned above, the continuous state space is split into regions so that for each raw state belonging to the same region executing the same action "feels" similar to the robot. This means that $Q(s, a, s')$ as the expected value for transitioning from s to s' with the greedy action $a = \pi(s)$ can be estimated with a sufficient confidence. This is calculated using interaction sequences starting in s and arriving in s' while only executing the greedy action a:

$$Q(s, a, s') = \gamma(s, a, s')\,(r(s, a, s') + V(s')) + \lambda(s, a, s')\rho(s, a, s') \tag{5.23}$$

Let $succ_a(s) = \{s' \mid P(s' \mid s, a) > 0\}$ be all successor states reachable from state s by action a. If raw states are mistakenly grouped into the same abstract region the variance of the values for $Q(s, a, s')$ calculated for all the greedy traces belonging to the same region will increase. A high variance of the experienced values for $Q(s,a,s')$ indicates that splitting that region will likely lead to better transition estimates in the split regions:

$$Var(\{\, Q(s,a,s') \mid s' \in succ_a(s) \,\}) > \theta_{TV} \tag{5.24}$$

This can be done by clustering the traces so that traces with similar $Q(s,a,s')$ are grouped together. For each cluster, one region is created.

The challenge is the determination of θ_{TV}. AMPS, which also uses a splitting heuristic based on the variance for $Q(s,a,s')$, requires the designer to analyze the scenario and empirically determine that value beforehand. This is apparently no possibility for groups of robots, which have to learn the proper behavior autonomously. As the distribution for Q usually cannot be foreseen it occurs that θ_{TV} is either too low, which results in too fine state abstraction and slows down the learning speed; or it is too high, which leaves too much aliasing in the strategy. The competing forces for determining θ_{TV} are therefore as follows:

1. The more often the robot is experiencing aliasing and the higher the variance of the values of the resulting regions' is, the higher the inclination to split should be.
2. The lower the variance is compared to the maximum region value the lower the inclination to split should be.

Both points are solved by replacing Var in Eq. (5.24) with $QVar$, which is defined in Eq. (5.25).

$$QVar(Q(s,a,s')) \equiv \sum_{s' \in succ_a(s)} P(s' \mid s,a) \frac{\left(\overline{Q(s,a,s')} - Q(s,a,s')\right)^2}{{V_{max}^{abs}}^2} \tag{5.25}$$

It normalizes the quadratic deviation of the Q values from their mean value, $\overline{Q(s,a,s')}$, to the maximal absolute region value $V_{max}^{abs} = \max(\{|V(s)| \mid s \in \mathbb{S}\})$. This is then weighted by the transition probability of the region. Since $QVar$ stays in the fixed interval $[0,1]$, the threshold θ_{TV} can be set to a fixed value independent of the future development of the region values. The inclination to split is thus adapting with the changing value function at runtime. A region is then split if the following condition holds:

$$QVar(\{\, Q(s,a,s') \mid s' \in succ_a(s) \,\}) > \theta_{TV} \tag{5.26}$$

5.5.2 Failure heuristic

With each region, a failure rate is associated. It describes the ratio of failure signals received when the greedy action of the corresponding region has been executed to the number of success signals. These signals are emitted by the strategy and skill layer, which will be described later. They are encoded as f_t in each interaction of the experience stream. Failure signals are scenario specific and can be emitted if, e. g., the robot bumps into a wall or if it has not encountered something interesting over a longer period of time. The failure heuristic splits a region if the failure rate of its greedy action is not sufficiently homogeneous. This is indicated by the condition

$$\theta_f < f < 1 - \theta_f\,, \tag{5.27}$$

with $0 < \theta_f < 1/2$. When decreasing the user defined threshold θ_f, the failure heuristic becomes more eager to split a region. This forces the state abstraction to attain a set of regions that have failure rates, which result in a more deterministic strategy. For both resulting new regions individual greedy actions can then be determined by the reinforcement learning algorithm.

5.5.3 Reward heuristic

Especially in the beginning of the robot's lifetime, when there is not yet enough information for the transition and failure heuristics to adapt the state space based on sufficient statistical data, the reward heuristic is of importance. It allows a region $s \in \mathbb{S}$ to be split if the variance of the reward rate is too high. This indicates that the action performed in that region receives a too diverse feedback in terms of the reward rates from the environment. A split of that region will then lead to multiple regions, which are more likely to be consistent with regard to the expected reward rates. This is also vital in cases where the failure signal is too seldom, as it provides the only other possibility to initially split a region.

In particular, the reward heuristic is looking at the reward rates of the experience stream for a clear switch from low to high variance areas, where both areas are of sufficient length. Such a switch in variance indicates that a split is advisable, as the robot experiences significant differences in the reward rates when executing the same action in the same region. Therefore, the reward heuristic considers the window of the last n reward rates made in the current region. The transition rewards in that time frame are not considered, as they only will show non-zero values in rare occasions. Let $\rho_{t-n}^{t} = (\rho_{t-n}, \ldots, \rho_t)$ and t be the current time. The reward heuristic is searching for an index k that splits ρ_{t-n}^{t} into the two sequences ρ_{t-n}^{t-k-1} and ρ_{t-k}^{t}, such that the following condition holds:

$$\begin{gathered} |\rho_{t-n}^{t-k-1}| > \theta_l \ \wedge\ |\rho_{t-k}^{t}| > \theta_l \\ \wedge \\ \big(Var(\rho_{t-n}^{t-k-1}) \approx 0 \ \wedge\ Var(\rho_{t-k}^{t}) > \theta_{RV} \ \vee \\ Var(\rho_{t-n}^{t-k-1}) > \theta_{RV} \ \wedge\ Var(\rho_{t-k}^{t}) \approx 0\big) \end{gathered} \tag{5.28}$$

The first part ensures that the split reward rate components are of sufficient length (θ_l). This is necessary for robustness against outliers in the reward rate stream. The second part tests for the switch from low to high variance regions and for high to low variance regions, respectively.

The minimum variance threshold θ_{RV} is dependent on the design of the motivation system. Recall from Eq. (5.11), Sec. 5.3, that the reward, which is received by the motivation system, is interpreted as a reward rate, if $|\dot{\mu}| \leq \theta_R$. With $\theta_{RV} = k \cdot \theta_R$, $(0 < k < 1)$, a switch is easily detected by the reward heuristic. The minimum sequence length of the individual variance subsequences, θ_l, ensures that the reward heuristic ignores trivial splits. Naturally it is set to be a fraction of the considered time horizon n. The detailed algorithm is provided in Alg. 1, p. 140.

5.5.4 Simplification heuristic

As splitting might lead to overly complex models, a means is needed that remerges regions once the robot has gathered new experiences that suggest a simpler model. This is achieved by the simplification heuristic, which analyzes sequences of regions connected by greedy actions. Similar to AMPS, the simplification heuristic considers *chain* and *sibling* merges. If a behaves nearly deterministically in s, the reachable successors are then denoted by $succ(s, a)$:

$$succ(s,a) \equiv \begin{cases} s', & \text{if } P(s'|s,a) \approx 1 \\ \text{None}, & \text{otherwise} \end{cases} \tag{5.29}$$

A *chain merge* of two regions s' and s' is performed if

$$succ(s', \pi(s')) = s'' \ \wedge \ succ(s'', \pi(s'')) = s \ \wedge \ \pi(s') = \pi(s'') \ . \tag{5.30}$$

In this case, the region s'' is superficial and can thus be merged with s' into the new region $s''' = s' \cup s''$, with $succ(s'''', \pi(s''')) = s$ and $\pi(s''') = \pi(s') = \pi(s'')$. All other regions that resulted into either s' or s'' are updated accordingly.

In the same vein, a *sibling merge* is triggered if

$$succ(s', \pi(s')) = s \ \wedge \ succ(s'', \pi(s'')) = s \ \wedge \ \pi(s') = \pi(s'') \ . \tag{5.31}$$

In this case, s' and s'' have similar expectations about the future region if the same action is executed.

5.5.5 Experience heuristic

This heuristic limits the memory horizon of the robot to $M_\theta \in \mathbb{N}^+$ interactions. It removes interactions that are too far in the past in order to keep the robot's model and policy aligned to the recent experience of the robot. Basically, it removes those old interactions from its memory and adds the new experience to it. Thus, it is modifying the experience of at most two regions which might cause an update of the model and of the policy.

5.6 Sample frequency

In order to let the chosen action take effect, the strategy layer is not triggering an action each time new perception is available. Instead, a new action is only triggered if at least one of the following conditions hold:

- The new perception differs sufficiently from the old one, measured by some scenario-specific distance metric d:
  $d(o_{t_1}, o_{t_2}) > \theta_o$

- The motivation layer has signaled a sufficiently interesting motivation change:
  $|\mu_{t_2} - \mu_{t_1}| > \theta_r$
- A certain amount of time has passed:
  $t_2 - t_1 > \theta_t$

θ_o, θ_r, and θ_t are application specific and have to be determined empirically. This dynamic sample frequency is also necessary for realistic applications to ensure that the robot is not overwhelmed by uninteresting information.

5.7 Exploration

With no information provided by the environment, the robot has to fall back to random exploration to actively request further information. As Whitehead notes, totally uninformed exploration is not likely to yield reasonable behavior, though [186]:

Learning is more often a transfer than a discovery. Similarly, intelligent robots cannot be expected to learn complex real-world tasks in isolation by trial and error alone.

Given, e. g., a one-dimensional grid with the states $s \in \mathbb{S} = \{-10, -9, \ldots, 10\}$, where the robot is located at $s = 0$ and where it has to reach one of the goals 10 or -10 by choosing actions from $A \in \{left, right\}$ with equal probability. As this is a discrete one-dimensional random walk, the average number of actions required to reach one of the goals without any further knowledge would be 100. And this applies just to the very simple grid example.

Two possibilities to overcome the problem of *tabula rasa* exploration within the reinforcement learning context are *guiding* [112, 166, 71] and *reward shaping* [120, 119, 135]. While *guiding* requires a form of teacher that provides salient knowledge and thus leads to a supervised learning setting, *reward shaping* only restructures the reward function in order to provide more instantaneous information to the robot. Thereby, the learning setting is still unsupervised. The restructured reward function then simply helps the robot to direct its exploration efforts. Reward shaping can be done manually beforehand [149, 108] or automatically online [117]. The previously described motivation layer has been designed in a way that allows it to be used as such a shaped reward function.

Practically, the robot is learning concurrently at the strategy and skill layer. One has to make sure that at any given point in time the other layer remains constant from the perspective of the learning layer [121]. From the perspective of the strategy learning, this can be assured using so-called GLIE policies (*Greedy in the Limit with Infinite Exploration*) [113]. A GLIE policy has the following characterization:

1. Each action is executed infinitely often in each state, which is also visited an infinite number of times.
2. The learning policy is greedy in the limit with respect to the value function. That means that the exploration rate is always positive, but decreases with time.

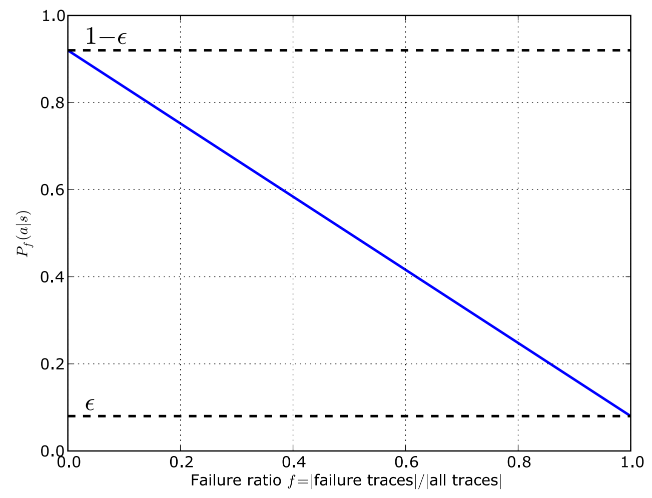


Figure 5.4: The probability of choosing action a in state s given its failure rate $f = |\text{failure traces}|/|\text{all traces}|$

The *Boltzmann exploration* $P(a\,|\,s)$ is an example that shows these properties. It calculates the probability of choosing action a in state s depending on some temperature parameter τ:

$$P(a\,|\,s) = \frac{e^{Q(s,a)/\tau}}{\sum\limits_{a' \in A} e^{Q(s,a')/\tau}} \tag{5.32}$$

When applied to the calculation of the action selection possibility in learning problems, τ starts with a high value and is "cooled down" with increasing experience. Along with this development the variance in the Q-values is becoming increasingly important, as high-valued actions are increasingly preferred over under-performing ones. In literature, τ is usually decreased as time goes by. It is not, however, the case that a robot gathers important experience as time goes by. As a consequence, a robot might settle on a policy after some time, even though the robot performed nothing useful at all. This might lead to sub-optimal strategies.

Therefore, it is advisable to use a replacement for $P(a\,|\,s)$ that better captures the notion of the robot's experience. The failure rate f is a candidate in this case. It is delivered by the skill layer and captures the experience in terms of whether the executed skill behaved according to the expectations. Fig. 5.4 shows the probability of choosing action a in state s dependent on the failure rate of that action: $P_f(a\,|\,s)$. It is parametrized by ϵ, which specifies the probability of random actions in the border cases $f = 0$ and $f = 1$ and determines the probability interpolation of $f \in (0, 1)$.

The question, when to explore at the strategy layer and when at the skill layer is answered heuristically in this thesis. Upon start, the strategy layer signals the skill layer to start its exploration phase. The strategy layer retracts the control, if the skill layer has signalled that it has learned the skills that are necessary for the scenario in question. Subsequently, the strategy layer explores according to $P_f(a\,|\,s)$.

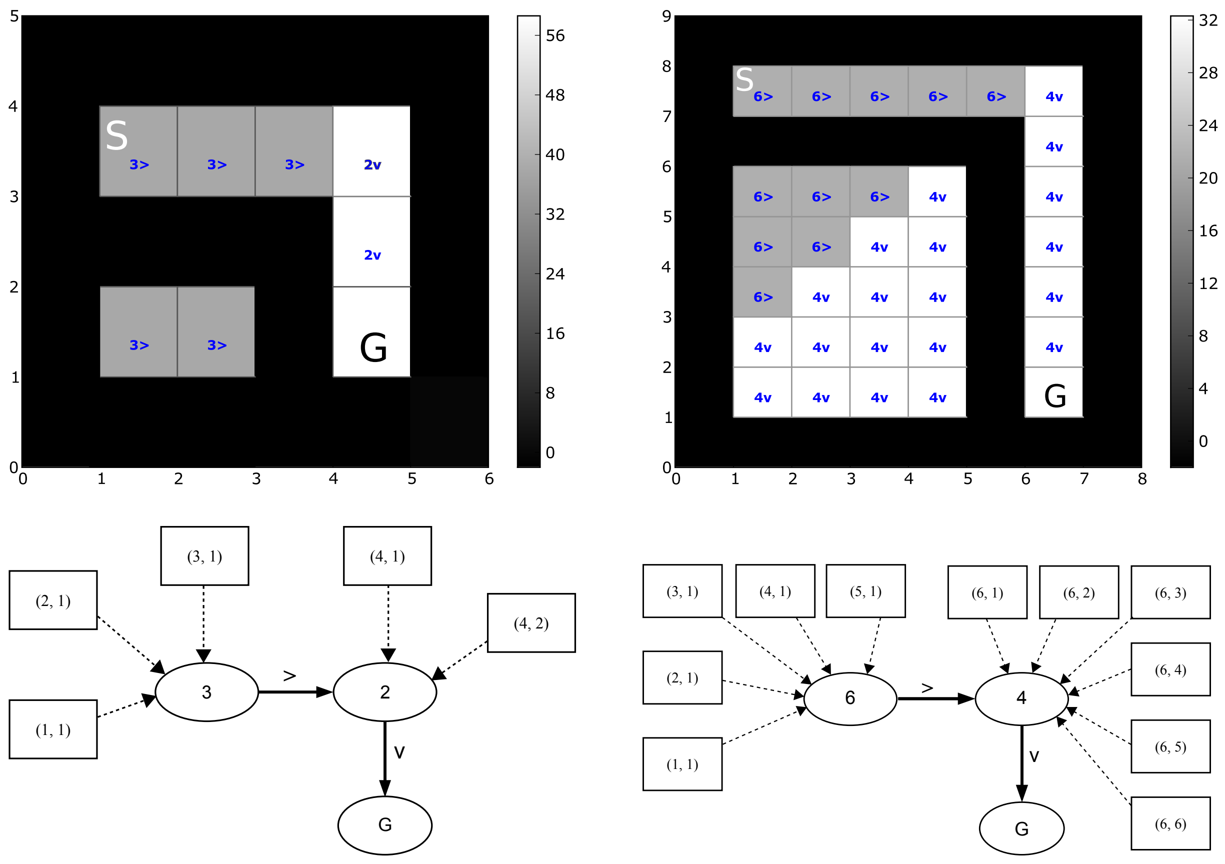


Figure 5.5: Upper figures: Two similar L-shape mazes of different size. The robot starts in the upper left ("S") and has to reach the lower right ("G") with the actions *north*, *east*, *south*, and *west*. The region number, its greedy action $\pi^*(s)$, and the region's value $V^*(s)$ (the brighter the higher) is shown for each state. **Lower figures:** Comparison of the strategies for the two mazes and the corresponding state abstraction mapping.

5.8 Example

To demonstrate how the strategy is able to extract the salient information to create an appropriate policy with the according state abstraction, two grid worlds of different sizes are presented in Fig. 5.5. The robot starts in the upper left and has to reach the lower right by choosing among the four possible actions *west*, *east*, *north*, and *south*[4]. The optimal strategy is the same in both worlds. As can be seen, the robot finds this same strategy for both worlds. The figure also shows how the approach separates states it has never encountered before into abstract regions. This is the case in the lower left corner in both grid worlds, which the robot has obviously never visited. In the right grid world, e. g., the area is diagonally split into the regions 6 and 4, based on the employed nearest neighbor heuristic.

So far, it has been assumed that $\mathbb{A}$ is always provided beforehand and that the strategy simply has to choose the right action at each state. For real-world scenarios it would be advantageous if $\mathbb{A}$ also could be learned at runtime. AMPS does this by applying similar abstraction heuristics to $\mathbb{A}$

[4]How more realistic continuous actions are learned and used will be described in the next chapter.

that helped to organize the state space $\mathbb{S}$. The actions learned in this way, however, are limited to simple domains, where the real-world dynamics can be presented by simple hypotheses. In the next section the skill layer is presented, which is able to learn reactive actions that are robust to noise and can handle complex dynamics.

CHAPTER 6

Skill layer

So far, the actions determined at the strategy layer are mere symbols, not able to work in the real world. To have any effect, they must be translated into low-level actuator commands. The bridge between these symbolic actions and the low-level actuator commands of the robot's hardware is the skill layer (Fig. 6.1) [7, 15, 21][1]. Its purpose is twofold:

- At the beginning, the skill layer has to autonomously learn a set of skills that are useful for the strategy layer. Thereby, the skill layer is *grounding* the symbolic actions of the strategy layer.
- During normal execution, the skill layer shall optimize its skills over the whole lifetime of the robot.

The skill layer perceives its environment in terms of features of objects like the relative distance to an object. Skills are only considered useful if they directly impact those features. If, e. g., a skill manages to decrease the feature *distance* to an object, it is considered interesting for the strategy layer and will be explored. The skill layer notifies the strategy layer about a new skill by sending the skill's identifier and the involved number of *target objects* if it considers that potential skill to be reliable. It will be described later on how this is measured by means of the skill's reproducibility.

In the following, *action* denotes the symbolic representation of a skill at the strategy layer, and *skill* the according representation in the skill layer that performs the action. A skill is a tuple of functions on the perceived features.

In order to increase the skill layer's robustness, it is not prescribed at design time which learning method to use at runtime. Instead, one is allowed to provide multiple possible learning methods.

[1]This approach has been implemented within the scope of the master thesis [180].

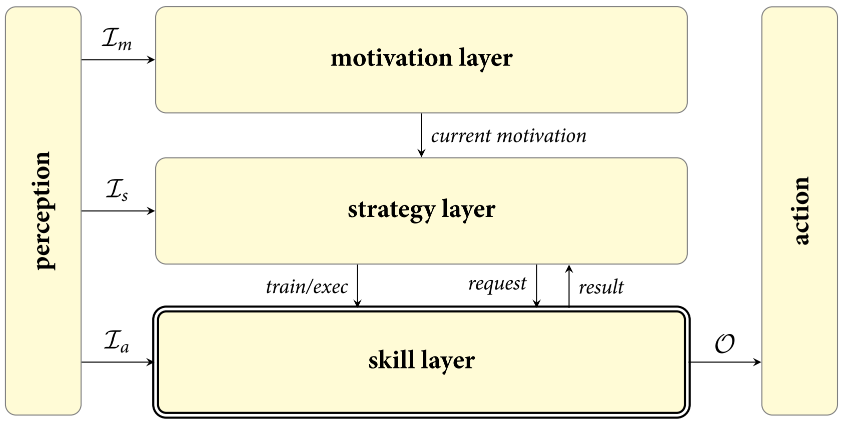


Figure 6.1: The layered robot architecture

The skill layer dynamically chooses the learning method that provides the best performance at runtime. This is achieved by continuously evaluating the learning methods through assigning them positive real valued scores. When the skill layer is executing a skill and has to choose between two or more competing models (which will be described later on) it chooses the one with the higher score.

6.1 Two modes of operation

When learning behavior at two different layers, the exploration-exploitation dilemma – naturally found in learning problems – is amplified. On the one hand, the strategy layer has to explore strategies over actions that are assumed to reliably yield the same result over time. On the other hand, the skill layer has to explore skills in order to provide the strategy layer with a sufficient set of usable skills.

This is solved by the strategy layer deciding when the skill layer is allowed to explore. If the skill layer is exploring, the strategy does not interfere by commanding, which skill to execute next. It waits for the skill layer to signal new skills that it deems to be reliable. The strategy layer updates its own action space each time accordingly. It triggers the skill layer to be in exploitation mode again if it deems its current action space as sufficient. From then on, the strategy layer is again in control and allowed to request the skill layer to execute skills. This leads to the two different operation modes: the *exploration* and *exploitation* mode. In the following, the data flows of both modes are described. The detailed description of the participating components will be given subsequently.

6.1.1 Exploration mode

In exploration mode (Fig. 6.2), the skill manager explores possible actions by generating potential skills and directly setting the actuator commands. At first, this is done randomly. It is called *motor babbling* as it refers to the early stages in child development, where the infant is exploring

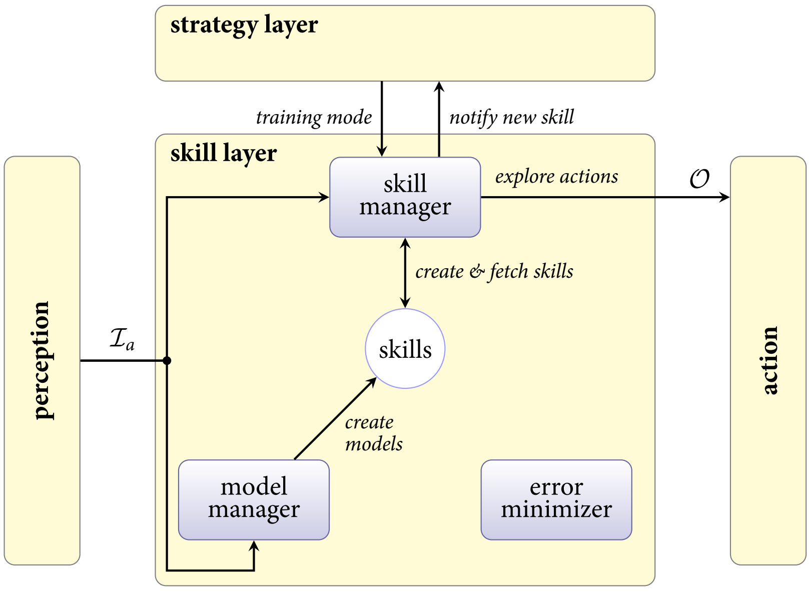


Figure 6.2: Data flow in exploration mode

its own sensori-motor coupling. While interacting with the environment, the model manager creates the corresponding models that predict the environment's and its own behavior. Over time, the predictions get increasingly more accurate until the skill manager regards some of the skills as reliable. In this case, it notifies the strategy layer, which in turn updates its action space.

6.1.2 Exploitation mode

In exploitation mode (Fig. 6.3), the strategy layer requests from time to time the skill layer to execute a skill by sending its identifier and a list of object IDs. From then on, the skill layer executes the skill by applying it to the specified objects. This allows for more generic skills at the skill layer. While the skill layer is executing a certain skill, it is still continuously updating its models, thus optimizing them all the time.

In this mode, the skill manager occasionally receives information from the strategy layer about the next skill to execute. This information is passed to the error minimizer, which from then on retrieves the corresponding models and ascertains the best possible actuator command for the perception at each time step. In parallel, the model manager is continuously updating the models with the new experience it receives at each time step with $\mathcal{I}_a$. While the basic behavior of a skill is fix from the strategy layer's point of view, the model manager is tuning them so that they increase in accuracy.

6.1.3 Interface with the environment

The skill layer expects the perception to be in factored or feature-based form. The skill layer's perception space $\mathcal{I}_a = \{(id_o, id_p, v) \mid id_o \in ID_o, id_p \in ID_p, v \in \mathbb{R}\}$ is a set of triples, each of

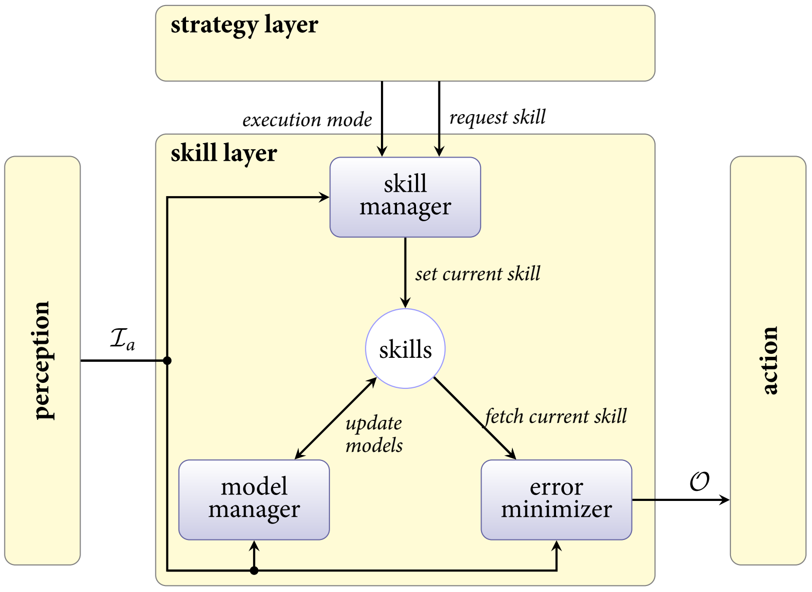


Figure 6.3: Data flow in exploitation mode

which describes the value v of an object id_o's perceptual feature id_p. ID_o and ID_p are the sets of the environment's objects and perceptual features. The perceptual feature, in the following also simply called *feature*, describes one specific aspect of an object and is chosen from a predefined list of possible properties ID_p. One example for such a set is $ID_p = \{angle, distance, color\}$. id_o is uniquely identifying one object in the environment. A typical example is the perception of the ball's relative position by means of its relative angle and distance:

$$I(t) = \{(ball, angle, 95.4), (ball, distance, 10.3)\}$$

Throughout this section the skill layer's input at time t is denoted by $I(t) \in \mathcal{I}_a$.

By relying on the perception to be in factored form containing object identities and properties, the skill layer is able to apply the same skill to different objects having the same property. If, e. g., the skill to approach the ball is learned by means of minimizing the perceptual feature *distance*, it can be applied also to approach the goal base or even to stay between the ball and the goal. In all cases the distance is minimized; in case of staying between the ball and the goal, the distances to both objects are minimized.

At each time step t, the skill layer determines a motor action $M(t) \in \mathcal{O} = \mathbb{R}^k$, where k is the number of actuators the robot is allowed to control. It contains one element for each actuator.

6.2 Component description

In this section, the components skill manager, model manager, and error minimizer are described. As they all rely on the skill repository, this will be defined beforehand.

The strategy layer activates a skill by sending the skill identifier to the skill manager. Internally, the skill layer retrieves the corresponding tuple of error functions from the central skill repository ("skills" in Fig. 6.3 and 6.2), which describe different aspects of the overall goal of that skill, and executes the corresponding low-level behaviors connected to those. At each time step, the skill layer monitors the current state of execution and may signal to the strategy layer one of the signals *success*, *failure*, or none at all.

For the definition of a skill, the notion of an *extraction*, *control*, and *error function* have to be introduced. Subsequently, the *progress function* will be defined, which calculates the overall progress state of a skill by means of the error functions in F_e.

Definition 6.1 *An* ***extraction function*** $f_{ext} : \mathcal{I}_a \rightarrow \mathbb{R}$ *extracts information from a perception* $I(t) \in \mathcal{I}_a$.

An extraction function can be implemented by simply returning one of the perceptual feature values from ID_p of a target object. It can be also more complex so that it returns the result of a mathematical operation on a list of the results of extraction functions.

Example 6.1 *Let* $\mathcal{I}_a = \{\{ball, goal\} \times \{angle, distance\} \times \mathbb{R}\}$ *and at time t be the perception* $I(t) = \{(ball, angle, 10), (ball, distance, 0.5), (goal, angle, 90), (goal, distance, 2)\}$. *If* f^1_{ext} *is implemented so that it returns* 0.5 *as the ball distance from the perception* $I(t)$ *and* f^2_{ext} *accordingly* 2 *for the goal distance, the previously mentioned behavior of staying between the goal and the ball will rely on the accumulated distances returned by* $f^3_{ext}(I(t)) = f^1_{ext}(I(t)) + f^2_{ext}(I(t)) = 2.5$.

By means of extraction functions, it is possible to select the desired information, which together with the following control function allows the specification of the behavior to be learned.

Definition 6.2 *A* ***control function*** $f_c : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$ *associates an error value to the tuple* (v_{t_i}, v_{t_j}), *where* v_{t_i} *and* v_{t_j} *are the values returned by an extraction function at different points in time* ($t_i < t_j$).

Example 6.2 *The following control functions are typical examples for specifying behavior, which decreases or increases a perceived property, respectively:*

$$\textit{control function to decrease value} : f_c(v_{t_i}, v_{t_j}) = |v_{t_j}| \tag{6.1}$$

$$\textit{control function to increase value} : f_c(v_{t_i}, v_{t_j}) = \frac{1}{|v_{t_j}|} \tag{6.2}$$

Both functions ignore the first argument. In Eq. (6.1), *e. g., the expression* $|v_{t_j}|$ *gives a higher error value the higher the current value* v_{t_j} *is. This has the effect that in order to return a low error value the control function needs the value to be decreased.*

Forcing a value of a perceptual feature to be decreased by a specific value δ *can be done with the following control function:*

$$\textit{control function to keep value} : f_c(v_{t_i}, v_{t_j}) = |v_{t_i} - \delta - v_{t_j}| \tag{6.3}$$

If, e. g., the ball distance at time t_i *is* $10m$, *and the robot shall approach it by* $\delta = 2m$, *the current error value at time* t_j *is defined by* $|10 - 2 - v_{t_j}|$, *assuming that* v_{t_i} *and* v_{t_j} *return the ball distance from* $I(t_i)$ *and* $I(t_j)$, *respectively. If the robot at time* t_j *is at a distance of* $8m$ *to the ball, the error is zero.*

Definition 6.3 *An **error function** $f_e : \mathcal{I}_a \times \mathcal{I}_a \to \mathbb{R}^+$ assigns an error value to the perception pair $(I(t_i), I(t_j))$:*

$$f_e(I(t_i), I(t_j)) = f_c\left(f_{ext}(I(t_i)), f_{ext}(I(t_j))\right) \tag{6.4}$$

$I(t_j)$ is the current perception and $I(t_i)$ the perception of f_e's first application, which is used as a reference point ($I(t_i), I(t_j) \in \mathcal{I}_a$).

Definition 6.4 *A **skill** is a tuple $s = (f_e^1, \ldots, f_e^N)$ of N error functions ($N \in \mathbb{N}$), which describe different aspects of the goal of the desired behavior. The tuple also serves as a unique identifier, by which the strategy layer requests this behavior together with N objects in corresponding order, to which the error functions will be applied.*

Example 6.3 *For a behavior of approaching the ball so that it finally is located directly in front of the robot, a skill s would be defined as follows:*

$f_{ext}^1(I(t))$	*returns the ball distance*
$f_{ext}^2(I(t))$	*returns the ball angle*
$f_c(v_{t_i}, v_{t_j}) = \lvert v_{t_j} \rvert$	*control function to minimize the value*
$f_e^1(I(t_i), I(t_j)) = f_c(f_{ext}^1(I(t_i)), f_{ext}^1(I(t_j)))$	*to minimize the ball distance*
$f_e^2(I(t_i), I(t_j)) = f_c(f_{ext}^2(I(t_i)), f_{ext}^2(I(t_j)))$	*to minimize the ball angle*
$s = (f_e^1, f_e^2)$	*skill to approach the ball and orient towards it*

The skill's overall progress is finally measured by a *progress function*. The robot can use it to both learn an according skill and recognize the skill in an observation, as will be described in Chap. 8. Thereby, the robot is technically emulating the mirror neuron system's behavior recognition as it is found in humans and animals [150]: the same neurons that are firing when a certain behavior is executed are also firing if the behavior is observed at someone else. In this thesis, the skill layer is learning to satisfy the progress function. The skill's same progress function is also used in the imitation phase to recognize the corresponding skill in observations.

Definition 6.5 *A **progress function** $f_p : \mathcal{I}_a \times \mathcal{I}_a \to [0, 1]$ is measuring a skill's progress between two time points t_i and t_j based on the perception at those points ($t_i < t_j$). A progress function is defined by two thresholds, $C_s \in \mathbb{R}^+$ and $C_a \in \mathbb{R}^+$ ($C_s < C_a$). For a skill $s = (f_e^1, \ldots, f_e^N)$ it is defined as*

$$f_p(I(t_i), I(t_j)) = \begin{cases} 0 & \text{if } C_a \le W(I(t_i), I(t_j)) \\ \frac{C_a - W(I(t_i), I(t_j))}{C_a - C_s} & \text{if } C_s < W(I(t_i), I(t_j)) < C_a \\ 1 & \text{if } W(I(t_i), I(t_j)) \le C_s \end{cases}, \tag{6.5}$$

where $I(t_i)$ is the perception when the skill has been started, $I(t_j)$ is the current perception, and $W(I(t_i), I(t_j)) = \sum_{k=1}^{N} f_e^k(I(t_i), I(t_j))$.

In Fig. 6.4, the progress function is plotted exemplary for $C_s = 0.15$ and $C_a = 0.75$, which are called *success* and *abort threshold*. The skill is considered as failed and can be aborted if the sum of the error functions, $W(I(t_i), I(t_j))$, exceeds C_a. In the interval $[C_s, C_a]$, the skill is considered to

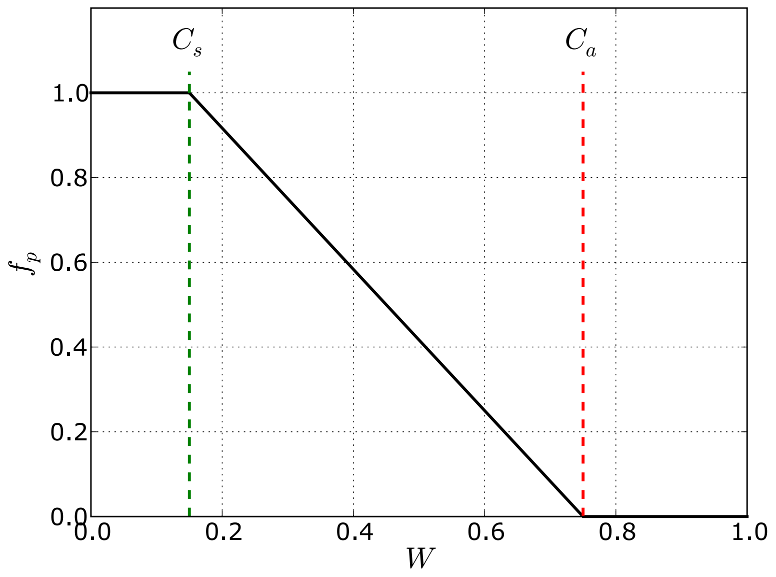


Figure 6.4: The progress function f_p plotted over the error function sum W (Eq. (6.5))

be normally executed. If $W(I(t_i), I(t_j)) > C_s$, the skill is considered to be successfully finished. These three states of the skill are measured by the progress function in the exploitation phase and used by the skill manager (Fig. 6.3), as will be described later on.

The thresholds C_s and C_a are determined by the skill manager in exploration mode (Fig. 6.2). For that purpose, the skill layer is monitoring the extremal values for the extraction functions, on which W is indirectly dependent through its error functions. The abort and success thresholds are then the values of W calculated from the error functions with extremal values for the extraction functions decreased and increased by a small tolerance, respectively. In case that the abort or success condition is met, the skill manager will signal the corresponding event to the strategy layer so that it can react accordingly.

The remaining chapter describes how the three components skill manager, model manager, and error minimizer interact with each other.

6.2.1 Skill manager

The skills of the skill layer are maintained by the skill manager. Its concrete tasks are to

- generate skills that enable the robot to control the perceived properties,
- assign a priority to each skill dependent on its execution priority,
- determine the skills the robot can reliably perform and notify them as new skills to the strategy layer, and
- manage the execution of requested skills.

The first three tasks are handled in the exploration phase, the latter one in the exploitation phase.

6.2.1.1 Skill generation

The skill generation naturally depends on the expressiveness of the skill definition, as described in Sec. 6.2. The more expressive it is in terms of possible perceivable features, extraction and control functions, the more possibilities the skill manager has to explore. In the evaluation experiments, the skill definition has been chosen in a way that lets the skill manager explore all possible skill definitions. For more complex scenarios where this is not possible, so-called feature selection mechanisms can be used to tackle this challenge. This is, however, outside the scope of this thesis.

6.2.1.2 Skill ranking

As it might be unfeasible to train all possible skills, the skill manager maintains a training priority for the potential skills. This is determined in training mode by a measure of interestingness for each skill. While the robot is moving randomly through the environment, it analyzes the perception stream to find interesting data in it. Let v_t be a value of a perception item $(id_o, id_p, v_t) \in I(t)$ at time t and $\boldsymbol{v}_{t_1}^{t_n} = (v_{t_1}, \ldots, v_{t_n})$. The interestingness event occurs if v_t either differs significantly from its n past values or if it is a new extremal value:

$$\begin{gathered} Var(\boldsymbol{v}_{t_1}^{t_n}) \ll Var(\boldsymbol{v}_{t_1}^{t_{n+1}}) \\ \bigvee \\ v_{t_{n+1}} < \min_{i=1,\ldots,n} \{v_{t_i}\} \vee v_{t_{n+1}} > \max_{i=1,\ldots,n} \{v_{t_i}\} \end{gathered} \tag{6.6}$$

Each time this condition is found in the current perception, the ranking of each skill s, $ranking(s)$, that impacts the corresponding object-feature tuple (id_o, id_p) by one of its error functions is increased.

As the skills' training is prioritised according to their ranking, this heuristic motivates the system to explore actions that control the most changeable properties. This is exemplary shown in Fig. 6.5 for the value v_t of two skills, s_1 and s_2, both of which impact the corresponding perception triple (id_o, id_p, v_t). s_1 is ranked lower than s_2, because some value v of a property causes less interestingness events. Training prioritization by means of the function $ranking(s)$ is a simple and fast heuristic to let the robot spend its time most useful. It will not waste its time by exploring potential skills that probably will not provide enough information during the exploration phase to create a useful skill.

6.2.1.3 Skill notification

The skill's reliability heavily depends on the designer's original goals concerning the robot. In some cases it is necessary to have very reliable actions at the cost of a longer training phase. In other cases it might be useful to have first sufficient actions available quickly at the beginning.

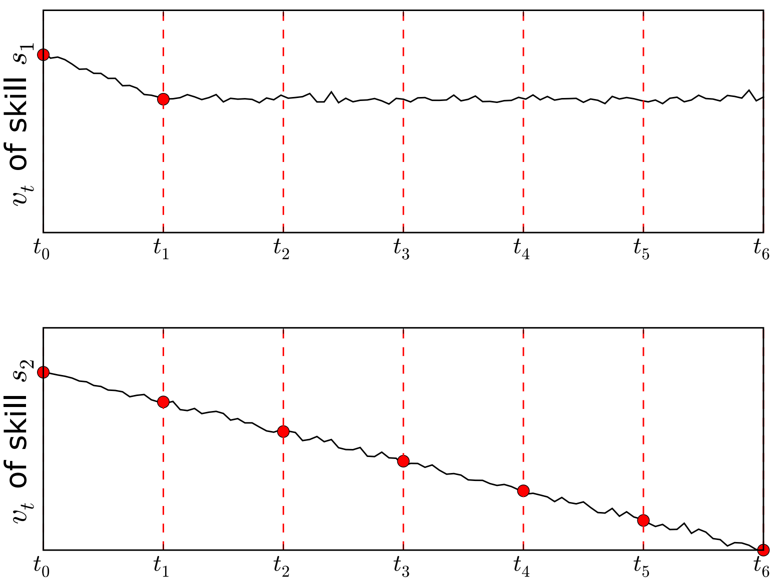


Figure 6.5: The two skills s_1 and s_2 are ranked differently due to the different amount of events. Interestingness events are marked with circles for each time step. As $ranking(s_2) > ranking(s_1)$, s_2 has a higher priority for exploration when the robot is in training mode than s_1.

After a skill has been successfully executed by a predefined number of consecutive times, the skill manager regards it as reliable and notifies it to the strategy layer, which can execute the skill henceforth.

Recall from Def. 6.4 that a skill is a tuple $(f_e^1, \dots, f_e^N)$ of error functions. When the skill layer notifies the strategy layer about a newly learned skill, the skill is bounded by the skill layer towards concrete objects.

Let $\boldsymbol{O}$ be the set of visible objects in the environment. The strategy layer updates its action space $\mathbb{A}$ with the new skill $(f_e^1, \dots, f_e^N)$ matched against the proper allocation of objects $(o^1, \dots, o^N) \in \boldsymbol{O}^N$ by

$$\mathbb{A} \leftarrow \mathbb{A} \cup \{\langle (f_e^1, o^1), \dots, (f_e^N, o^N) \rangle \mid (f_e^i, o^i) \neq (f_e^j, o^j) \; \forall i \neq j \wedge o^k \in \boldsymbol{O}\} . \tag{6.7}$$

Thereby, unnecessary actions like $\langle (\textit{minimize angle}, \textit{ball}), (\textit{minimize angle}, \textit{ball}) \rangle$ can be avoided.

On the strategy's side, $\langle (f_e^1, o^1), \dots, (f_e^N, o^N) \rangle$ serves no other purpose than uniquely associating an action in the strategy's action space with the corresponding skill in the skill layer. Thus, the strategy layer can treat its actions as symbols, which are grounded by the skill layer. *Symbol grounding* [86] is the process in robotics research that is equivalent to the sensorimotor stage in Piaget's theory on child development [139]. This way, the strategy layer can concentrate on sequences of actions while the skill layer is handling the execution of the actions in form of skills.

6.2.2 Model manager

The environment's reaction to a low-level action is described by prediction models. Given the current perception, a set of prediction models describe how the perception will change after having executed the skill for one time step. The model manager is in charge of maintaining a sufficient set of prediction models. It does so by

- creating prediction models for each perceived property,
- updating prediction models to reflect new experiences, and
- calculating a score for each model dependent on its prediction accuracy.

The creation of models takes place in the exploration phase, while the updating and continuous scoring of prediction models is carried out in the exploitation phase.

Definition 6.6 *A **prediction model** is defined by the tuple $(id_p, \tilde{S}, \tilde{M}, m)$. $id_p \in ID_p$ is the perception feature to be predicted. $\tilde{S} \subset ID_o \times ID_p$ is a subset of the perceptual features of perceived objects. $\tilde{M} \subset \mathcal{O}$ is a subset of the actuators the model should incorporate. The prediction function $m\colon \mathbb{R}^{|\tilde{S}|+|\tilde{M}|} \to \mathbb{R}$ predicts the value for the perceptual feature id_p at the next input perception given the values of $\tilde{S}$ and $\tilde{M}$.*

6.2.2.1 Creating and updating models

Because of its freedom in definition, the model manager is challenged with a very high number of possible combinations of $\tilde{S}$, $\tilde{M}$, and regression algorithms when creating models in the exploration phase. It has to determine which subset $\tilde{S}$ of the perceived properties, which subset $\tilde{M}$ of the motor signals, and which regression algorithm to use.

For each skill, multiple prediction models compete to be used based on their past prediction accuracy. This is measured by the squared error loss function

$$L = (Y - m(X))^2 \,,$$

which is to be minimized in the fitting process of the prediction models. This allows to provide the skill layer with a number of different function approximators for m. It will continuously test them to find out at runtime, which one is the best predictor.

Throughout this thesis, the skill layer is provided with the radial basis function approximation (RBF) for high generalization and polynomial approximation for fast calculation as the modeling algorithms. Concrete definitions and examples will be given in Sec. 7.3. However, the skill layer is not dependent on these two specific approximation techniques. In fact, any function type can be used for m, as long as it supports the necessary input and output spaces.

6.2.2.2 Scoring models

The training data for the models is generated as follows. At each time t, the current perception $S(t)$ is determined by the previous perception $S(t-1)$ and by the last low-level action $M(t-1)$. Therefore, a new experience tuple $(S(t-1), M(t-1), S(t))$ is available at each time step.

Each time a new feature is perceived, the model manager generates a set of models that predict the feature's value using diverse function approximators and input spaces. Each of these models is continuously updated with new experiences that activate at least one *interest signal*. The model manager disposes of two such signals, the *surprise* and *mistake* signal.

surprise signal The surprise signal is triggered if the prediction error is more then twice the average prediction error of the model.

mistake signal The mistake signal is activated if the sign of the property's value change predicted by the model is different from the real one.

These signals are inspired by the principle that humans learn when there is a discrepancy between what happens and what they actually have predicted that should have happened [184].

Finally, the model manager computes the score of each prediction model function m as the inverse of the mean squared error in predicting the last n interactions with the environment:

$$score(m) = \frac{n}{\sum_{i=k}^{k+n} \left(m(S(t_i), M(t_i)) - v_{t_{i+1}}\right)^2} \tag{6.8}$$

The model with the highest score, m_{best}, is used as the prediction model for the skill.

6.2.3 Error minimizer

In the exploration phase (Fig. 6.3), the error minimizer uses the prediction models learned by the model manager to determine the best next motor vector $M(t+1)$ to send to the actuator for the skill requested by the strategy layer based on the current perception. The error minimizer minimizes the error functions f_e^i of the current skill $s = (f_e^1, \dots, f_e^N)$ for the next perception, which is called the *expected next error* $e(t+1)$. It can be computed as a function of the low level action M described by the following algorithm.

1. Determine the perception $I_c(t)$ that contains only perceptual features, on which the error functions of the current skill s are dependent:

$$I_c(t) = \left\{(id_o, id_p, v) \mid (id_o, id_p, v) \in I(t) \ \wedge \ id_p \in ID_p^s\right\} , \tag{6.9}$$

where ID_p^s is the set of all perceptual features the skill function depends on. This can be either directly by way of the extraction functions (Def. 6.1), on which the skill's error functions f_e^i depend (Def. 6.3), or indirectly through the perceptual features, on which the skill's prediction models depend (Def. 6.6).

2. Determine the best actuator command M_{best} for skill s. This is the actuator command that yields the minimal error expected by the prediction models according to the skill's error functions:

   (a) Estimate the next perception, $I_c(t+1)$, dependent on the motor action M as predicted by m^j_{best}:
   $$I^M_c(t+1) = \left\{ m^j_{best}(I_c(t), M(t)) \mid p_j \in I_c(t) \right\} \tag{6.10}$$
   The prediction model m^j_{best} is the one with the highest score for predicting p_j, as maintained by the model manager:
   $$m^j_{best} = \arg\max_m \{score(m)\} \tag{6.11}$$

   (b) Calculate the expected next error $e^M_k(t+1)$, with $I_c(t_i)$ being the perception when the skill has been started:
   $$e^M_k(t+1) = f^k_e(I_c(t_i), I^M_c(t+1)) \tag{6.12}$$

   (c) Determine the best actuator command $M(t)$, by finding the one that minimizes the accumulated expected error:
   $$M(t) = \min_M \sum_{k=1}^{N} e^M_k(t+1) \tag{6.13}$$

$M(t)$ is found in a two stage process. At first, a coarse grid is projected into the low-level actuator space. From the actuator commands at the grid points the actuator command $M_{coarse}(t)$ with the lowest predicted error is determined. Starting with $M_{coarse}(t)$, constrained optimization by linear approximation [141] (COBYLA) is used to determine the approximate optimal one. It is approximated as the process is used with a timeout in order to timely deliver an actuator command. In the optimization process, COBYLA's time complexity is determined by the prediction model function $m_{best} : \mathbb{R}^{|\tilde{S}|+|\tilde{M}|} \to \mathbb{R}$ (cf. Def. 6.6). As $I_c(t)$ does not change during that process, the approach reduces computation costs by generating a sub-model function $m'_{best} : \mathbb{R}^{|\tilde{M}|} \to \mathbb{R}$ with fixed input, and uses that instead of m_{best} in the calculation of $M(t)$.

6.3 Configuration

As already pointed out, the greater universality leads to a bigger exploration space. Although the skill layer is fully autonomous and able to cope with that, it is wise to limit the exploration space by specifying non-changing parameters beforehand. This can be achieved by configuring the following parameters:

- *Degrees of freedom* specify the number of actors the skill layer has to control.
- *Extraction functions* define the language that can be used to specify the error functions.

- *Control functions* specify the functions that the error minimizer will minimize by means of the error functions.
- *Regression models* are used by the model manager to build predictions for the environment interaction. A regression model consists of two methods: one that fits a model to an experience trace and one that predicts the value of the modeled property.

6.4 Conclusion

The skill layer finds out by itself what types of capabilities are actually learnable before it starts trying to learn specific skills. The learned skills are then adapted while being executed. The main point of this skill layer, however, is its ability to detect behavior in observation streams. With this capability, the robot can ask the individually learned skills, whether there are behavior patterns in the observation that could also be achieved by the skill itself. With this information, the imitation approach of Chap. 8 is then able to abstract and recognize complex behavior in the observation.

CHAPTER 7

An integrative example

This chapter shows exemplary how all three layers can be specified so that a robot learns to collect objects and carry them to a goal in a Capture-The-Flag (CTF) scenario. As already discussed in Sec. 1.1, CTF is often used as a canonical task to evaluate multi-robot systems and can be seen as a model for real-world tasks like dirt cleanup, search and rescue operations, and similar tasks. The autonomously learned behavior will be demonstrated and tested within the Player-Stage/Gazebo [79] simulation (Fig. 7.1). The Pioneer2DX robot serves as the robotic platform. It has four wheels and a gripper, which is used to push the object. The dynamics are simulated using the *Open Dynamics Engine (ODE)* [168].

The scenario consists of a goal base to which objects, which are dispersed in the environment, have to be transported. The robot has to find out which skills, which have to be autonomously learned by the skill layer (Chap. 6), have to be executed in which order, learned by the strategy layer (Chap. 5), to achieve that goal. Although the scenario is quite simple, it shows all the characteristics of real-world scenarios, i. e., it is noisy, continuous, and time-dependent.

The results regarding the strategy layer are averaged over 200 experiments, in which the robot had to push an object 30 times consecutively to the goal. The confidence interval of 95% is provided. The charts regarding the skill layer are individual examples.

7.1 Implementation of the motivation layer

For this example it suffices to equip the robot with only one drive. A positive transition reward of 100 is given if the robot has pushed the object to the yellow goal base. The change of the distance between the nearest object and the goal is provided as reward rates. The perception $\mathcal{I}$ is preprocessed to provide the robot's relative distance d_g to the goal g and the robot's relative

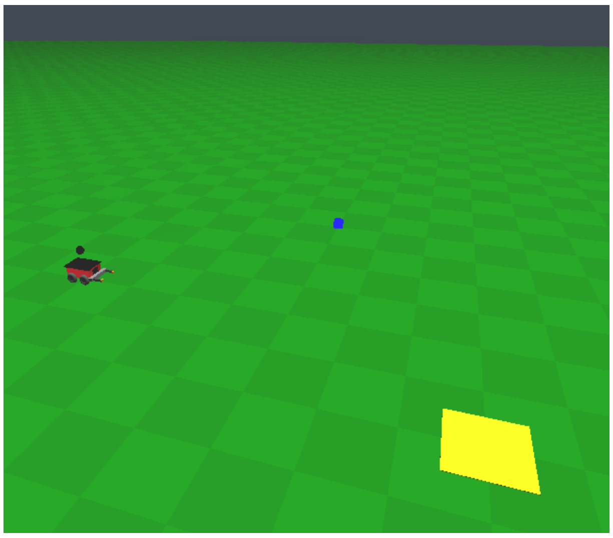


Figure 7.1: Capture-The-Flag scenario. The robot has to learn to push the blue object to the yellow goal base. It has to learn by itself both the low-level actions and the strategy using them, involving the proper state abstraction and the correct timing of the actions

distance d_q and angle α_q to the object q:

$$\mathcal{I}_m = (d_g, d_q, \alpha_q) \in \mathbb{R}^3 \tag{7.1}$$

The motivation layer was defined to provide the strategy layer with the following motivation:

$$\mu_1(\mathcal{I}_m(t)) = \begin{cases} 0 & \text{if } d_g < 1m \wedge d_q < 0.5 \wedge |\alpha_q| < 20^\circ \\ \max\left\{0, \mu_1(\mathcal{I}_m(t-1)) - \frac{\Delta d_q/\Delta t}{100}\right\} & \text{if } |\alpha_q| < 20^\circ \\ \max\left\{0, \mu_1(\mathcal{I}_m((t-1)) + 0.01\right\} & \text{otherwise} \end{cases} \tag{7.2}$$

This provides a high reward for reaching the goal ($d_g < 1m$) with the object in the gripper ($d_q < 0.5 \wedge |\alpha_q| < 20^\circ$), because it is setting the motivation to zero. If the object is far away but in front of the robot ($|\alpha_q| < 20^\circ$), it gets the small incentive $-\frac{\Delta d_q/\Delta t}{100}$, which is the bigger the nearer the object gets. Otherwise, the motivation is increased by 0.01. This forces the robot to prefer faster strategies.

With this definition, the robot will receive different rewards for reaching the goal, based on how high its motivation grew in the past. This is, however, not of a problem for the strategy layer, as it treats the received rewards as statistical samples.

7.2 Implementation of the strategy layer

The strategy's state space consists of the robot's relative distance and angle to goal g and object q:

$$\mathcal{I}_s = (d_g, \alpha_g, d_q, \alpha_q) \in \mathbb{R}^4 \tag{7.3}$$

As described in Chap. 5, the task of the strategy layer is to generate a strategy that guides the robot through this state space in order to satisfy the motivation layer. Based on the current state of $\mathcal{I}_s$, the robot chooses the best action according to that strategy. It has been discussed that this is infeasible with the original state space of $\mathcal{I}_s$ as it would take too long for the robot to find a usable strategy. Instead, the strategy layer abstracts the continuous 4-dimensional state space into a manageable amount of abstract states. These abstract states, also called regions, are determined by means of the heuristics presented in Sec. 5.5. A region consists of states that "feel similar" to the robot when executing the same action. This means that the failure rates, durations, and expected rewards are similar for all the states of the same region given the same action.

For this to work properly, the adaptation heuristics of the state space, which are described in Sec. 5.5, have to be parametrized. The choice of the parameters is domain dependent and has to be done manually using expert knowledge of the domain. The following parameters were therefore determined empirically:

- *Transition heuristic:* It determines whether a region s should be split because the region s' that is reached by the greedy action $\pi(s)$ has a too low probability $P(s' \mid s, a)$. The heuristic is controlled by the threshold θ_{TV} in Eq. (5.26) (page 47). The higher this threshold the more tolerant the strategy layer is regarding greedy next states, which are ambiguous. With this threshold the strategy layer trades off policy accuracy against the number of involved regions. A lower ambiguity is bought by a higher number of regions.

  The threshold is determined to be $\theta_{TV} = 0.2$. If a region's greedy action has no clear next region as a result, it will be quickly split with this value.

- *Experience heuristic:* This heuristic controls the robot's memory. By limiting the number of the robot's past experiences, it basically forgets experiences that are too old. If an old experience drops out of the memory, the policy is updated to account for the changed memory. Without this heuristic, the complexity would rise continuously as an increasing amount of experiences would have to be handled by the other heuristics.

  In this chapter, the number of experiences is not bounded ($\theta_M = \infty$), but stays below 9,000 (Fig. 7.2).

- *Failure heuristic:* It adjusts regions based on the failure signals received by the skill layer. While the robot is in normal execution mode, the skill layer may send failure signals meaning that the applied skill did not perform as expected. Due to the noise in the environment, the skill layer may signal a failure even if there is no problem in the environment. In that case, the failure signals should be ignored. If, however, the failure signals arrive at a high frequency, this is a sign that the state space may not be properly abstracted. This means that skills make assumptions about the outcome of their application that don't hold in the given state. The failure heuristic splits the region corresponding to that state so that the policy can better distinguish the different states. Eq. (5.27) (page 47) determines whether the corresponding region is split, based on the failure threshold θ_f.

  In the experiments, $\theta_f = 0.01$ ensures that regions are split at very low failure rates.

- *Reward heuristic:* In the beginning of a robot's lifetime, there is usually not enough information on which the transition and failure heuristics can base their state splitting decisions. In that case, the strategy layer contains just one region into which all observed states are mapped. In this situation, the reward heuristic comes to the rescue. It analyzes the observed reward rates in order to find a significant change in the reward rate variance as defined by Eq. (5.28) (page 48). The point in the experience stream, where this is the case might be a good choice for a split. All experienced states that belong to the same region before that point are separated from those that come later. The time window in which the reward heuristic looks for such a variance change is defined by n. The minimum reward rate variance is specified by θ_{RV}. The splitting is, in addition, only performed if the traces with low and high variance both contain at least θ_l state observations.

The strategy layer considers the reward rates of the last $n = 20$ interactions and used the constants $\theta_{RV} = 0.01$ and $\theta_l = 6$.

- *Simplification heuristic:* The application of the abovementioned heuristics results in an increasingly detailed state abstraction. The simplification heuristic counteracts this development by merging regions that are overly discriminating in the light of new experience. This is specified by the definition of when an action is viewed deterministic, as described in Sec. 5.5.4.

  In the experiments, an action a in state s was considered deterministic, if $P(s' \mid s, a) \geq 0.8$ for some next state s'. Given the following exemplary policy with the corresponding transition probabilities:

$$\begin{array}{ll} \pi(s_1) = a_1 & P(s_2 \mid s_1, a_1) = 0.9 \\ \pi(s_2) = a_2 & P(s_3 \mid s_2, a_2) = 0.8 \end{array} \tag{7.4}$$

  In this case the two actions a_1 and a_2 are considered as being deterministic. As a consequence, the simplification heuristic will merge s_1 and s_2.

These heuristics are responsible for maintaining a reasonable state space abstraction. The policy that is built using those abstract states is governed by the discount factor β (cf. Sec. 5.1.2 and 5.3). With a value of β near zero, the robot is increasingly paying attention to a reward that is further in the future. At higher values of β the robot becomes increasingly near-sighted (Eq. (5.12), p. 43). The discount parameter of the strategy is set to $\beta = 0.1$.

7.3 Implementation of the skill layer

The skill layer is configured to control two degrees of freedom: velocity and rotational speed. It was provided with one control function "decrease" defined similarly to Eq. (6.1) (page 59):

$$f_c(v_{t_i}, v_{t_j}) = |v_{t_j}| \tag{7.5}$$

The skill layer preprocesses the perception $\mathcal{I}$ in order to provide the robot's relative distance and angle to the goal and to the object:

$$\mathcal{I}_a = \{\{ball, goal\} \times \{angle, distance\} \times \mathbb{R}\} \tag{7.6}$$

In this experiment, polynomials and radial basis functions (RBF) are used for m. The polynomial function can be calculated fast but has restricted generalization capabilities, while an RBF is time-consuming but offers much better generalization possibilities.

Polynomial The multivariate polynomial is defined by

$$m^{poly}(\boldsymbol{x}) = \sum_{d_1=0}^{d} \dots \sum_{d_n=0}^{d} c_{d_1,\dots,d_n} x_1^{d_1} \cdot \dots \cdot x_n^{d_n}\,, \tag{7.7}$$

where d is the degree of the polynomial and $c_{d_1 \dots d_n}$ are the coefficients.

E. g., the polynomial $m^{poly}(\boldsymbol{x}) = x_1 x_3^4 x_{10} + 3x_2^2 x_3 x_5^7$ is realized by the coefficients $c_{1,3,10} = 1$ and $c_{2,3,5} = 3$, where the first summand has $d_1 = d_{10} = 1$ and $d_3 = 4$, and the second has $d_2 = 2$, $d_3 = 1$, and $d_5 = 7$.

The degree d is determined using the early stopping technique. It is biased towards simpler models in that it increases the model complexity by means of its degree until no significant increase in accuracy can be observed.

Radial basis function An RBF is defined by

$$m^{rbf}(\boldsymbol{x}) = \sum_{i=1}^{N} \lambda_i \phi(\|\boldsymbol{x} - \boldsymbol{x_i}\|)\,. \tag{7.8}$$

$\| \cdot \|$ defines the Euclidean distance between the *center* $\boldsymbol{x_i}$ and a data point $\boldsymbol{x}$. ϕ is the multiquadric basis function $\phi(r) = \sqrt{(\frac{1}{\epsilon r})^2 + 1}$ with $\epsilon = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x_i}\|$. The final result is weighted by $\lambda_i \in \mathbb{R}$. They are computed by solving the set of linear equations $y_i = \lambda_i(\phi(\|\boldsymbol{x_i}\|) - \rho)\ \forall i \in \{1..N\}$, where ρ is a smoothing constant. An example of an RBF, which is used as a prediction model for a skill, is shown in Fig. 7.5 (p. 76).

The model manager is allowed to generate polynomials of maximal degree ten. The smoothing constant is set to $\rho = 0.001$ for the RBF model function to smooth it slightly.

7.4 Evaluation

The evaluation compares two settings. In the setting for the first type of experiments ("manual"), the motivation layer and strategy layer are used as defined above. The skill layer, however, consists of manually programmed skills. The second type ("learned") uses the skill layer as defined in Sec. 7.3 instead. Wherever appropriate, the charts contain a 95% confidence-interval.

Fig. 7.2 shows how the robot manages to abstract the 4-dimensional state space of the strategy layer into a small number of abstract regions, with which the strategy is learned. With hand-crafted skills it has abstracted the observed states of nearly 8,000 interactions into only seven abstract states. With everything being learned, meaning that also the skills are autonomously learned by the robot, the robot has generated only 13 regions out of more then 8,500 state observations. This shows that the robot separate the infinite 4-dimensional state space into a low

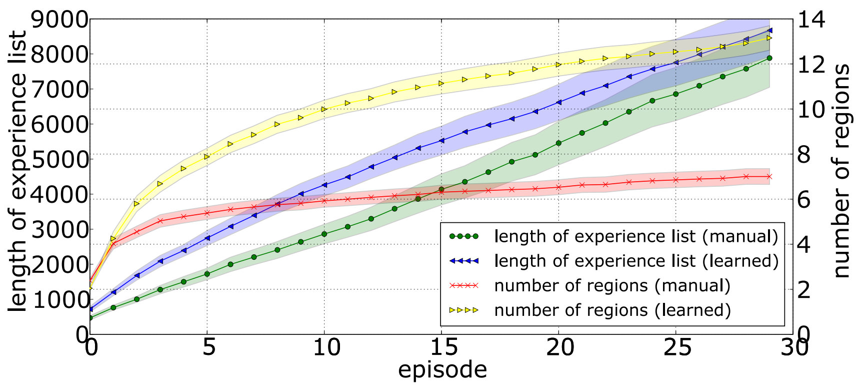


Figure 7.2: Size of experience and number of abstract regions

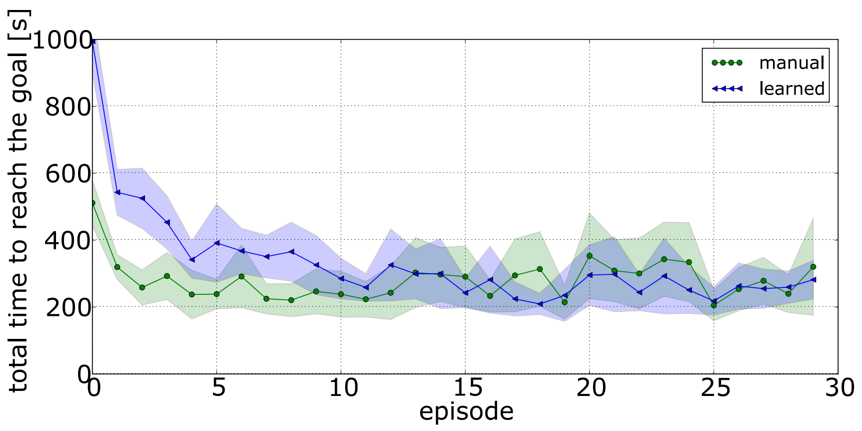


Figure 7.3: Time to push the object to the goal

number of abstract regions, by which it is able to explain the environment and its interactions with it.

Comparing the total time of the first run the robot needs for pushing the object to the goal, the learned version needs approximately 1, 000s compared to 500s in the manual version (Fig. 7.3). This is explained by the fact that the robot needs more exploration time when also learning at the skill layer. There it also has to explore its own capabilities and learn the necessary skills. Both charts converge slightly above 200*s*. It shows that while being slower in the beginning, the learning skill layer manages to finally converge to the same performance, which is nearly optimal for this scenario.

The reward per second is displayed in Fig. 7.4. The learning skill layer stays slightly below the manual version. The learning version, however, can be considered as far more robust. This shows that the approach is capable to autonomously tackle infinite state and action spaces in realistic scenarios.

The skill layer has autonomously generated different competing prediction models that determine the behavior of the learned skills in all experiment runs in the end. The skill layer, however, has always chosen RBF for the finally learned skill due to its greater accuracy. Only in the beginning of the experiments, when only little data is available, the polynomial is chosen. The learned

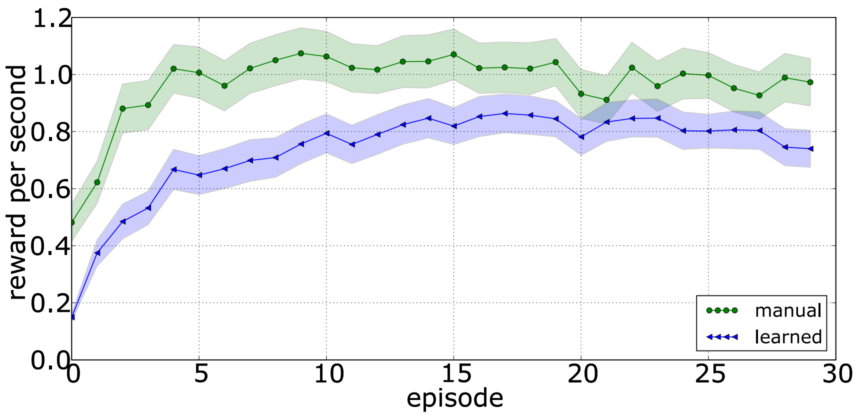


Figure 7.4: The reward per second

behavior will be represented in a synthetic way to show the proceedings of the skill layer (Fig. 7.5). The utilized prediction model is based on the radial basis function approximation. It predicts the next value of the angle by knowing the value of the angle and the distance to the object and the chosen low-level action. A grid of 30x30 points in the input space is used. The input dimensions are the angle and the distance to the object, so each point of the grid is characterized by a certain angle-distance couple. For each point of the grid, the error minimizer (Sec 6.2.3) computes the low-level action that minimizes the predicted distance. The result is one 3-dimensional graph for each degree of freedom: one indicating the chosen tangent speed and the other indicating the chosen rotational speed. The third dimension (the actuator intensity) is represented by colors: red for negative intensity, white for zero, and blue for positive intensity.

In Fig. 7.5, the behavior of decreasing the angle to the object is represented. The upper graph indicates the rotational speed dependent on the angle and distance to the object. For each angle-distance combination a full blue color means rotating right with highest rotational speed (100), and a full red color means rotating left (-100). The values between are interpolated accordingly. A white color, e. g., means no rotation. This is the case, if the angle is already minimized. Otherwise, it is set to turn towards the object. It can be seen that the robot correctly learns to set the rotational speed dependent on the relative angle to the object.

The frontal velocity is shown in the lower graph and looks more noisy than the rotational speed. It shows that the robot drives backwards when it is close to the object. In effect, driving forwards in that case could lead to a continuous rotation around the object that would never lead to a decrease of the distance. When the distance is not low, there is not a clear behavior with regard to the forward speed. This makes sense, because it does not much affect the angle to the object so it can even be chosen randomly without side effects on the performance of the action.

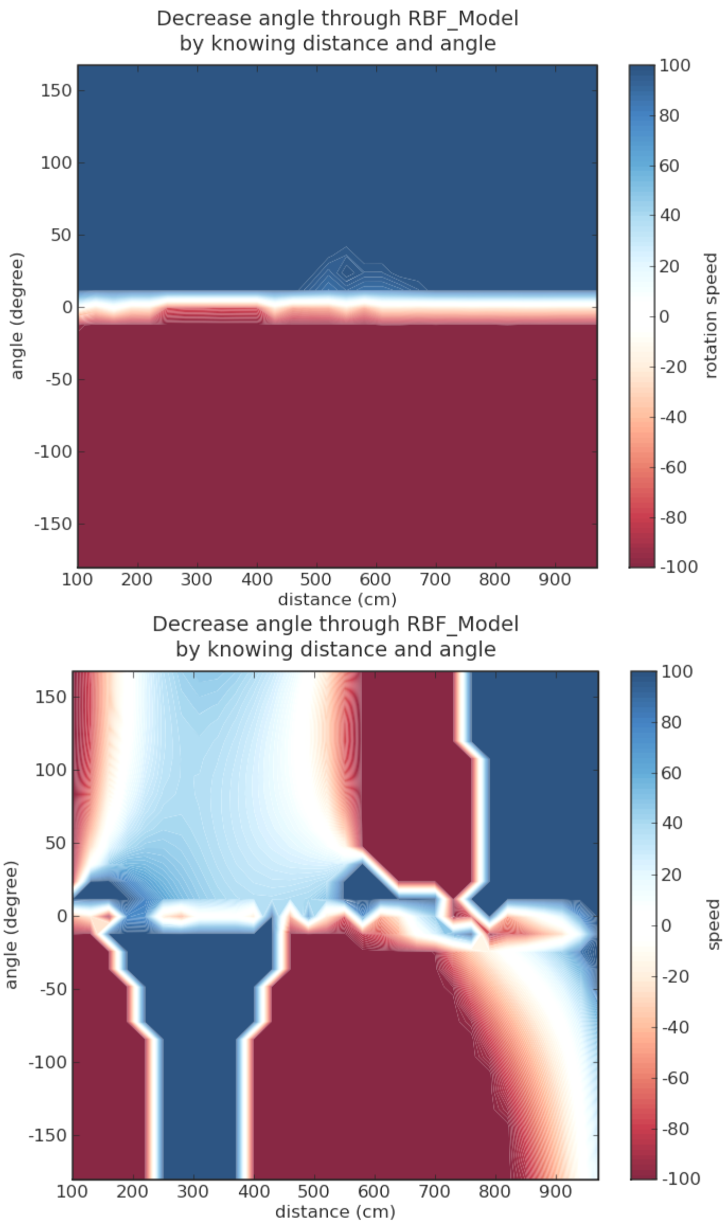


Figure 7.5: Low-level actions associated to the abstract action of minimizing the angle to the object. The red color denotes a full negative value (-100), while the blue one a full positive one (100).

CHAPTER 8

Imitation in robot groups

The previous chapter has laid out the basis for a robot to successfully bootstrap its own learning process. How a robot can speed up this process by imitating the other robots' successful behaviors will be presented in this chapter [18, 19, 20][1]. It comprises

- algorithms to recognize, interpret – and thereby understand – the observed behavior (Sec. 8.4),
- the integration of understood behavior into the robot's own knowledge (Sec. 8.5), and
- experimental results that display that this approach increases the learning speed and performance (Sec. 8.6).

Although the material regarding imitation in robot groups is rather sparse, some work has already been conducted in this field of research. Before presenting the approach of this thesis, the current approaches will therefore be presented and contrasted.

8.1 Related work

The field of imitation has already been overviewed in Chap. 2. This section will survey the state of the art in multi-robot imitation. In addition to the previously described challenges that the more constraint imitation approaches described above have to face, multi-robot imitation has to cope with the following problems:

- When to imitate (cf. Sec. 2.2.2.1).

[1]Parts of this approach have been implemented within the scope of the diploma thesis [159].

- How to integrate the observed behavior into the already individually learned behavior knowledge.

This section surveys the research that tackles especially these challenges in the context of multi-robot and multi-agent context.

Takahashi et al. use imitation to learn robotic soccer behaviors like approaching or shooting a ball [175, 174]. During imitation a robot in their approach uses the value function approximated to the observation of the imitatee to estimate the imitatee's performed policy. They rely on manually discretized state spaces. The approach requires the following set up: The imitatee first performs an action 10 times while the imitator observes it. Then, the imitator tries to perform the action with the observation knowledge. Afterwards the imitator evaluates the behavior, as well as its recognition performance. Finally, the procedure starts over again until the imitator has learned the action. This approach requires the robot group to stop in their current task whenever an imitator tries to learn new behavior by imitation.

Priesterjahn uses imitation in the field of multi-agent first-person shooter games [142, 143, 144, 145] to generate non-person characters (NPC) that maximize the human game player's fun. The difficulty in designing such games is not to maximize the NPCs game strength, but to keep the game agents at a strength level similar to the human player's. Priesterjahn achieves this by evolving neural network controllers, which incorporate successful behavior from other NPCs as well as behavior information from the human player. At first, a human player's actions are recorded while he is playing the computer game. The recorded data is then used as a starting rule base to evolve a neural network at play time. A rule is the mapping of an environmental state to an action. The environmental state is basically the grid map (30 × 30 cells in his experiments) with a value for each grid cell indicating whether the cell is free, blocked, or containing an opponent. The finally learned behavior consisting of the learned rules is thus targeted to that particular grid and player setting. Priesterjahn combines imitation with learning in that the NPCs are sharing so-called "elite" situation/action-rules. These are rules that performed exceptionally well during the game. This requires the situation, as well as the action, to behave similar for all NPCs. The action is a 4-tuple (forward movement, lateral movement, view angle, attack). The forward movement, e. g., allows for forward, no movement, and backward ({1,0,-1}). Thus, the approach is targeted to game worlds and cannot easily be transferred to real-world robotic scenarios. In addition, it requires all NPCs to commit to the same communication channel and protocol. In real-world applications, this type of imitation is problematic if the communication channel is too noisy or temporarily not working. In addition, his approach is designed for game scenarios that contain actors with the same behavior capabilities. The question, how to deal with heterogeneous groups is therefore not handled by his approach. It, however, showed that imitation-based adaptation is able to outperform the evolutionary only approach.

Closest to the approach of this thesis come Inamura et al. [93, 94] with their *Mimesis Loop* approach. Thereby they are able to symbolize observed low-level behavior traces. This is used as top-down teaching from the user's side in combination with the bottom-up learning from the robot's side. As this is useful to decrease the programming effort, it is an exclusive solution, not allowing to be used with other learning techniques like, e. g., reinforcement learning. Also their approach is not able to use already existing abstract states of the imitator in the recognition process. Once a robot has extracted enough information to construct a hidden Markov model

(HMM) based on the recognized low-level behaviors, it is fixed to that HMM – no exploratory actions are possible any more on the abstract states. Furthermore, the segmentation process that splits the continuous movement trajectories into basic movements uses a fixed scheme. With that it is not possible to allow for ambiguities at the recognition phase.

In summary, there is either work in the field of imitation that allows single robots to learn individual tasks from a predetermined teacher, and there is imitation employed in multi-agent situations that misses the action recognition and correspondence problem task. Furthermore, up to now no research has been carried out regarding sporadic imitation, which is apparently very important when robots in groups should benefit from each other's learning efforts. Typically, the imitation process should not interrupt the observed robot, so that the imitating robot often has only one example of interesting behavior to learn from. This usually does not provide enough information for learning a generalized skill that can be replayed later on. Instead of trying to learn low-level skills from observation, the observer nevertheless might use the observation to update its strategy if it is able to retrieve the salient information. How this can be achieved will be described in this chapter.

8.2 Overview of the multi-robot imitation approach

It lies within the nature of robot groups that the information retrieved during sporadic observation in an imitation process is rather sparse, since imitation possibilities do not show up very often. And even when there is such a possibility, the observing robot should not disturb the imitatee or demonstrator by requesting a repetition of the behavior observed previously. This renders the imitation of low-level skills unfeasible, because it usually requires a lot of data gathered by observation of many repetitions before the skill can be replayed satisfactorily.

Nevertheless, a robot group can benefit from sporadic imitation if its robots attempt to imitate the broader picture of the general behavior, i. e., the imitation of strategies or sequences of low-level skills. At this level of behavior, imitation provides enough information for the following exploitation possibilities of the observation:

1. The robot may decide to spend more exploration efforts on the state transitions just observed, which play a role in the sequence of newly observed behavior.
2. If the robot observes state transitions for which it cannot find already known skills in its skill repertoire, it can direct its exploration efforts of the skill layer to learn that specific possibly new skill.
3. The robot can incorporate the transition data of the observed demonstrator condensed into its own strategy.

The imitation approach presented in this chapter revolves around the third possibility, as the exploration process would exceed the scope of this thesis (cf. Sec. 8.5).

The general approach is shown exemplary in Fig. 8.1 where a robot (imitator) tries to *understand* the observed behavior episode of an other robot (demonstrator or imitatee). The episode consists

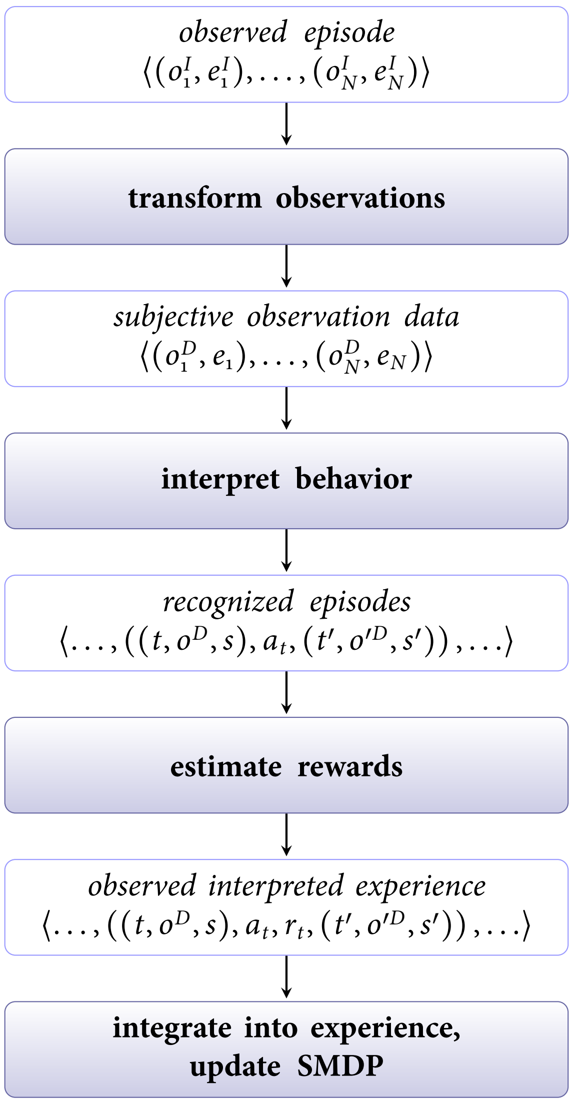

observed episode
$\langle (o_1^I, e_1^I), \ldots, (o_N^I, e_N^I) \rangle$
transform observations
subjective observation data
$\langle (o_1^D, e_1), \ldots, (o_N^D, e_N) \rangle$
interpret behavior
recognized episodes
$\langle \ldots, ((t, o^D, s), a_t, (t', o'^D, s')), \ldots \rangle$
estimate rewards
observed interpreted experience
$\langle \ldots, ((t, o^D, s), a_t, r_t, (t', o'^D, s')), \ldots \rangle$
integrate into experience,
update SMDP

Figure 8.1: The process of imitation: observing an other robot's behavior, interpreting it in terms of its own knowledge, and integrating it into the latter

of the raw perception as observation o^I and the visible "well-being" state e^I of the demonstrator D as observed by the imitator I (hence the superscript). The well-being state is comparable to an emotional state that comprises the robot's overall state in form of a drive state (Chap. 4). The robots permanently observe each other and maintain a window of predefined length of the last observations and well-being states. If a robot shows a significant change in its well-being state and an observing robot detects that, it will try to imitate only the section of the observed episode that contains the quasi-monotonic well-being state.

The observation o^I is subjective to the imitator I, consisting of perception data that contains the coordinates of the imitatee or demonstrator D, which is moving around. To really understand what the demonstrator did, the imitator has to "look with the demonstrator's eyes". That means that the imitator has to translate all the coordinate information in the observations to see what it would have perceived in the demonstrator's situation.

It then interprets the subjective perception by allowing its low-level skills $a \in \mathbb{A}$ to give so-called *votes* (cf. Chap. 6), which express how well each of them could have achieved the observation changes. Using an algorithm inspired by the Viterbi algorithm from the field of Hidden Markov Models (HMM) [148], those votes are then used in combination with the imitator's state space $\mathbb{S}$ in order to find the most likely behavior sequence corresponding to the observations. Subsequently, the recognized episodes are enriched with the estimated information for the missing data. Finally, this is integrated into the imitator's experience and the strategy is updated.

8.3 Transforming observations

Consider the example, where an imitator I observes a demonstrator D driving directly to a goal base. The demonstrator perceives its environment in terms of distance and angle to that goal: $\mathcal{I}_s = \mathbb{R}^2$. Over a period of time, the demonstrator perceives the following sequence for $\mathcal{I}_s$:

$$\langle \dots, (3m, 20^\circ), (2.7m, 17^\circ), \dots, (0.2m, 3^\circ) \rangle \tag{8.1}$$

As the demonstrator perceives the last state, $(0.2m, 3^\circ)$, the robot has reached its goal and receives a high positive reward from its motivation layer. As a consequence, the demonstrator's strategy layer updates its policy to account for the fact that it will receive positive reward with higher probability when the goal is near and in front of it.

The observing imitator I has, however, no access to the demonstrator's subjective perception. It has to infer it from its own observations. The raw state space of its strategy layer usually does not even contain enough information regarding other robot's coordinates as it is kept as small as possible for better learning convergence. The imitator instead has the ability to access the raw information available from its perception component before it is preprocessed, which is $\mathcal{I}$ (cf. Fig. 3.1, p. 19). It contains positional data including the 3-D position and orientation subjective to the imitator regarding all the objects and robots in the vicinity. Therefore, the imitator collects the following information when observing the demonstrator, where all of the positional data in the perceived observation is imitator centric:

$$O^I = \langle \mathcal{I}_1^I, \dots, \mathcal{I}_N^I \rangle \ , \tag{8.2}$$

where O^I contains the window of the last N received raw perceptions.

Let exemplary $\mathcal{I}_k^I = \mathcal{I}_k = \big((\boldsymbol{v}_g^I, \boldsymbol{\theta}_g^I), (\boldsymbol{v}_D^I, \boldsymbol{\theta}_D^I), c\big)$, $k \in \{1, \dots, N\}$, be defined as the 3-D positions and orientations of goal g and robot D and some other non-vector value c. $\boldsymbol{v}_g^I$ and $\boldsymbol{v}_D^I$ are then regarded as positional data. In the following, the superscript denotes the coordinate system of the positional data. E. g., $\boldsymbol{v}_g^I$ is a vector of the goal g in the coordination system of the imitator I. As the transformation is done with respect to D's coordinate system, $(\boldsymbol{v}_D^I, \boldsymbol{\theta}_D^I)$ will be replaced by the relative position and orientation of the imitator I as seen from the demonstrator D. All other positional data – in this case only $\boldsymbol{v}_g^I$ – will be converted from the imitator's coordinate system (I) into that of the demonstrator (D). The conversion makes use of D's relative position $\boldsymbol{v}_D^I$ and orientation $\boldsymbol{\theta}_D^I$ so that it reflects D's subjective observation.

This transformation is achieved by the mapping $\varphi_D : \boldsymbol{v}_O^I \mapsto \boldsymbol{v}_O^D$. It has to be applied to each positional data individually. Let $\boldsymbol{v}_D^I = (x_D^I, y_D^I, z_D^I)^T$ and $\boldsymbol{\theta}_D^I = (\alpha_D^I, \beta_D^I, \gamma_D^I)^T$ be retrieved from $\mathcal{I}$

as the 3-D position vector and orientation of the demonstrator's coordinate system in the imitator's coordinate system. Prior to the transformation they are converted to homogeneous coordinates [185]. Such a coordinate is a four-element column vector, where the first three elements are the original coordinates and the fourth is a scale factor, which is always 1 in this case. Homogeneous coordinates allow the translation to be represented as a matrix multiplication, which unifies all rotation and translation matrices into one. These homogeneous transformation matrices, namely the rotation around the three axes and the translation as shown in Fig. 8.2, are defined by Eq. (8.3) and (8.4).

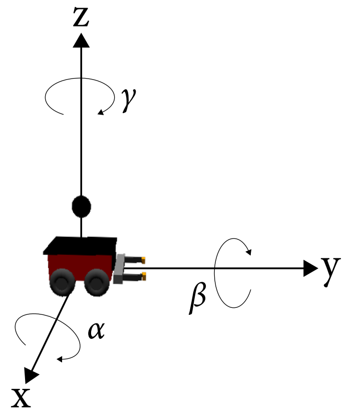


Figure 8.2: The robot's coordinate system

$$\boldsymbol{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha^I_D & -\sin\alpha^I_D & 0 \\ 0 & \sin\alpha^I_D & \cos\alpha^I_D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{R_y} = \begin{pmatrix} \cos\beta^I_D & 0 & \sin\beta^I_D & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta^I_D & 0 & \cos\beta^I_D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \boldsymbol{R_z} = \begin{pmatrix} \cos\gamma^I_D & -\sin\gamma^I_D & 0 & 0 \\ \sin\gamma^I_D & \cos\gamma^I_D & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8.3}$$

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & x^I_D \\ 0 & 1 & 0 & y^I_D \\ 0 & 0 & 1 & z^I_D \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8.4}$$

The chain multiplication $\boldsymbol{M_{D\to I}} = \boldsymbol{R_x R_y R_z T}$ transforms an object's position from the demonstrator's point of view into the point of view of the imitator. As the recognition algorithm needs the opposite effect, the inverse of $\boldsymbol{M_{D\to I}}$ results in the transformation matrix $\boldsymbol{M_{I\to D}}$, which converts an object from the coordinate system of the imitator to that of the demonstrator:

$$\boldsymbol{M_{I\to D}} = \left(\boldsymbol{R_x R_y R_z T}\right)^{-1} \tag{8.5}$$

In the frequent case that imitator and demonstrator are coplanar ($z^I_D = \alpha^I_D = \beta^I_D = 0$), the transformation may be simplified, because of $\boldsymbol{R_x} = \boldsymbol{R_y} = \boldsymbol{E}$. The transformation matrix $\boldsymbol{M_{I\to D}}$ transforms an object's position $\boldsymbol{v^I_O} = (x^I_O, y^I_O, z^I_O)^T$ in the imitator's coordinate system into that of the demonstrator's:

$$\boldsymbol{v^D_{O'}} = \boldsymbol{M_{I\to D}} \begin{pmatrix} x^I_O \\ y^I_O \\ z^I_O \\ 1 \end{pmatrix} \tag{8.6}$$

$\boldsymbol{v^D_{O'}}$ is a vector in homogeneous coordinates. The 3-D vector $\boldsymbol{v^D_O}$, which specifies the object's position subjective to the demonstrator, is retrieved by extracting the first three coordinates of $\boldsymbol{v^D_{O'}}$.

The last coordinate, being the scaling factor, may be ignored. With Eq. (8.6) the demonstrator is able to generate the according subjective observation for each observation for any observable demonstrator:

$$\langle \mathcal{I}_1^D, \dots, \mathcal{I}_N^D \rangle \tag{8.7}$$

This can then be preprocessed by the user-defined function $\Upsilon_s : \mathcal{I} \to \mathcal{I}_s$ (cf. Chap. 3) in order to retrieve the state in the strategy layer's state space, which is the perception that the demonstrator D has seen:

$$O^D = \langle \mathcal{I}_{s_1}^D, \dots, \mathcal{I}_{sN}^D \rangle \tag{8.8}$$

The recognition process can now analyze O^D in order to recognize behavior that probably has led to the positive reward.

8.4 Understanding observed behavior

The desired outcome of the observation and recognition phase in an imitation process is a state-action-trace that results in a performance similar to the observations. For this, the imitating robot has to find corresponding states in its own strategy's state space that might play a role when imitating the observed behavior. Furthermore, it should only regard states that can be connected by means of actions of which the imitating robot is capable. This requires the imitator to couple its strategy and skill layer, whereby it is able to accomplish the recognition and understanding of other robots' behavior in terms of its own strategy and skill capabilities. As described in Chap. 5, the strategy is modeled with a Semi-Markov Decision Process (SMDP) that has a dynamically adjusting state space. It uses self-developed skills as actions, which are triggered in terms of goal functions on the perception (Chap. 6).

With the described means for strategy and skill learning, the Viterbi algorithm can now be adapted accordingly. It is often used to replay behavior previously encoded by a Hidden Markov Model (HMM) [148] in an imitation process. HMMs are stochastic models that describe Markov processes. HMMs are often used in the field of imitation to encode sequential patterns of motion as stochastic finite state automata [52]. Before the recognition algorithm is presented, a short overview of the Viterbi algorithm following the description of Bengio [40] is given. Throughout this chapter the following notation is used:

- $P(o\,|\,s)$ is the likeliness of observing o in state s.
- $P(s'\,|\,s) = \sum_a P_a(s'\,|\,s)$ is the probability of s being the next state after s' independent of which action $a \in \mathbb{A}$ has been chosen.
- $P(o_t\,|\,o_{t-1}, a) \equiv P_a(o_t\,|\,o_{t-1})$ is the probability of observing o_t after having observed o_{t-1} in the previous time step and executed action a henceforward.
- $P(s'\,|\,s, a) \equiv P_a(s'\,|\,s)$ is the probability that action a executed in state s transitions to s' (cf. Sec. 5.3).
- $T(s, a, s')$ is the strategy's individually learned probability of transitioning to state s' when started in state s and executing action a henceforward (called $P(s'\,|\,s, a)$ in Sec. 5.3).

In the following, the time stamp subscript t refers to a continuous point in time at which the according attribute has been perceived. The continuous time point of the attribute in the previous time step is addressed by $t-1$.

8.4.1 Viterbi

The *Viterbi path* is usually calculated by the imitation approaches in the literature to find the state sequence that the imitator later on should realize in order to *exactly* copy the observed behavior [52]. This is carried out by using the state space of the inferred HMM, which is assumed to be fix and to reflect the demonstrator's state and action space together with the state transition probabilities.

The path is calculated by the Viterbi algorithm [182], which attempts to find the most likely hidden state sequence $s_1^N = \langle s_1, s_2, \ldots, s_N \rangle$, $s_i \in \mathbb{S}$, that best explains the observation sequence o_1^N, $o_i \in \mathbb{R}^d$ with d being the dimension of the observation vector. It is achieved by maximizing the probability $P(s_1^N \mid o_1^N)$:

$$s_1^{N*} = \arg\max_{s_1^N} P\left(s_1^N \mid o_1^N\right) \tag{8.9}$$

The Viterbi algorithm efficiently determines the maximum in time $O(Tn)$ using Bellman's dynamic programming algorithm, where n is the number of non-zero transition probabilities [39]. It does so by recursively calculating the probability[2]

$$\mathcal{V}(s,t) = \max_{s_1^{t-1}} P(o_1^t, s_1 \ldots s_{t-1} s_t = s) \tag{8.10}$$

that $s \in \mathbb{S}$ is the observed hidden state at time t given the observations o_1^t:

$$\mathcal{V}(s,t) = P(o_t \,|\, s_t = s) \max_{s'} \left[P(s_t = s \,|\, s_{t-1} = s')\mathcal{V}(s', t-1)\right] \tag{8.11}$$

With an initial assignment of $\mathcal{V}$ by

$$\mathcal{V}(s,1) = P(o_1 \,|\, s_1 = s) P(s_1 = s) \;\forall\; s \in \mathbb{S} \tag{8.12}$$

the most likely path can then be extracted with backward recursion:

$$\varphi(s,t) = \arg\max_{s'} \left[P(s_t = s \,|\, s_{t-1} = s')\mathcal{V}(s', t-1)\right] \tag{8.13}$$

It determines the best predecessor of state s at time t.

8.4.2 Interpreting observed behavior

The approach in this thesis utilizes cues from the Viterbi algorithm. In contrast to the approaches in the literature, the Viterbi algorithm in this thesis is merely used to explain the observations

[2] In order to not confuse the value of a hidden state in the Viterbi path with the value of a state in reinforcement learning (cf. Sec. 5.3), it is denoted by $\mathcal{V}$ instead of V.

recorded from the demonstrator with the current knowledge of the imitator. In this way, the imitator tries to *understand* the demonstrator with the knowledge it already has attained in terms of its own strategy knowledge (cf. Chap. 5) and skill repertoire (cf. Chap. 6). The presented algorithm takes as the input an episode $O = \langle o_1, \ldots, o_N \rangle$ consisting of demonstrator centric observations. It then extracts a list of understood state transitions of the form

$$\Gamma = (\ldots, ((t, o, s), a, (t', o', s')), \ldots) \ . \tag{8.14}$$

For this, Eq. (8.11) as part of the recognition part in the imitation process has to be modified accordingly.

Using the state abstraction mapping $\xi : \mathbb{R}^d \to \mathbb{S}$ from Sec. 5.4, $P(o_t \mid s_t = s)$ could easily be realized as an indicator function:

$$P(o_t \mid s_t = s) = \begin{cases} 1, & \text{if } \xi(o_t) = s \\ 0, & \text{otherwise} \end{cases} \tag{8.15}$$

This would, however, yield a weight, which is too high, to the imitating robot's current mapping of ξ, because it rules out all other states $\mathbb{S} \setminus \{\xi(o_t)\}$. At the other side of the extreme, ξ could be ignored by assigning the same probability to all observations in order to take all other abstract states into account. However, as the robots in the group are considered to have similar goals, the imitator should also exploit its current mapping assuming that the chosen demonstrator has a similar one. This leads to the trade-off discussed in the following.

As an example, consider Fig. 8.3 where the most likely state has to be determined for the observation o. As described in the previous chapter, the nearest neighbor approach is chosen throughout this thesis to map the state observations to the abstract states of the strategy. Let N_o^k be the set of k observations in the robot's experience that are nearest to o: $N_o^3 = \{o_1, o_2, o_3\}$. Furthermore, define $N_{o\to s}^k$ to be those observations in N_o^k that are mapped to s by ξ:

$$N_{o\to s}^k = \left\{ \hat{o} \mid \hat{o} \in N_o^k \ \wedge \ \xi(\hat{o}) = s \right\} \tag{8.16}$$

In the example figure, $N_{o\to s_2}^3 = \{o_1, o_2\}$. $P(o_t \mid s_t = s)$ can then be defined to be inversely dependent on the distance to the labeled observation instances in the nearest neighbor representation:

$$P(o_t \mid s_t = s) = \frac{\sum_{\hat{o}\in N_{o\to s}^k} \| \ o_t - \hat{o} \ \|^{-2}}{\sum_{\hat{o}\in N_o^k} \| \ o_t - \hat{o} \ \|^{-2}} \tag{8.17}$$

During the recognition phase, the robot thereby gives all regions their chance, depending on how much their observations resemble the one in question. In the case of $N_{o\to s}^k = N_o^k$, Eq. (8.17) behaves like the initial suggestion in Eq. (8.15) for $P(o_t \mid s_t = s)$.

The calculation of $P(s_t = s \mid s_{t-1} = s')$ in Eq. (8.11) is more involved. If one would just take the transition probability of its greedy action in s_{t-1}, $P(s \mid s', \pi(s'))$, the robot would not get new insights about other – and maybe better – state transitions in that specific state. Instead, it should guess from the observations which of the skills in its own skill repertoire would best realize the recorded observations.

Given an arbitrary state transition $\langle s_{t_1}, s_{t_2} \rangle$ with $s_{t_1} \neq s_{t_2}$ $(t_1 < t_2)$. For each recorded observation step $\langle o_{t-1}, o_t \rangle$, $[t-1, t] \subset [t_1, t_2]$, all the skills estimate their *vote* $P_a(o_t \mid o_{t-1})$. With this vote, the

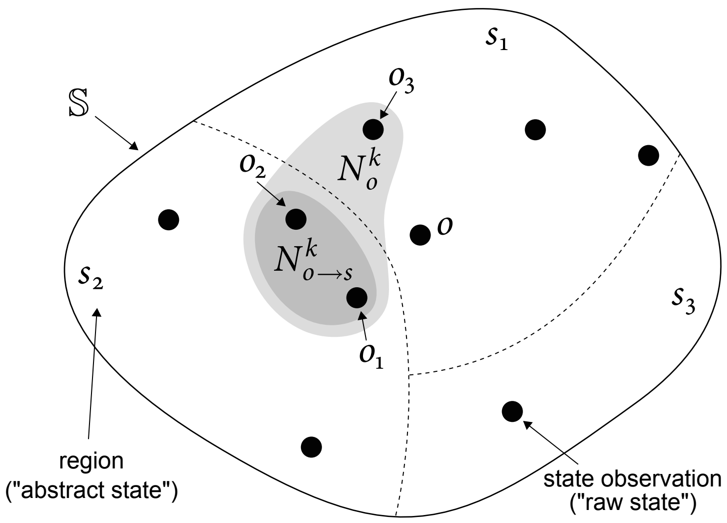


Figure 8.3: The calculation of $P(o_t \,|\, s_t = s)$ is based on the distances to the k nearest observations. In this example, N_o^k and $N_{o\rightarrow s}^k$ are shown for $o_t = o$, $s = s_2$, and $k = 3$.

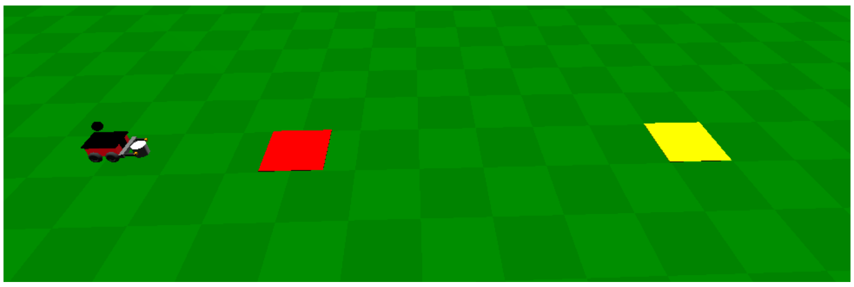


Figure 8.4: Example of a demonstrator that might drive either to the red (left) or the yellow (right) goal base. The heuristic assigns a higher probability to that skill that is nearer to its optimum (in this case, "drive to red base")

skill a estimates its ability to transition the robot in a way that realizes the change from observation o_{t-1} to o_t. It does so by means of the corresponding progress function f_p, with which the skills were learned (Chap. 6). The vote calculation is governed by the following heuristics:

- The skill delivers a higher vote if its progress function is nearer to its optimum. As a consequence, the skill's attention is focused to nearer objects, which are considered as more relevant to the robot's actions. This is necessary to filter out unrelated skills. As an example, Fig. 8.4 shows a robot, which is pushing an object towards two possible goals. For an observing imitator robot having two skills a_{red} and a_{yellow}, both would vote equally as it is ambiguous to which goal the demonstrator is actually driving. With this heuristic, a_{red} will vote higher then a_{yellow}, because the robot is nearer to the red goal.

- If the progress function is within a given tolerance range ϵ_a around the optimal value 1 it delivers a vote of zero. This is necessary since small but irrelevant changes of the progress functions of skills that have already accomplished their tasks will result in votes that rule out other skills that are currently working towards their goals. ϵ_a can be derived while skill

a was learned: it is dependent on the overall variance while executing the skill.

- The vote is clipped to the interval $[0, 1]$. Negative votes occur when $f_p^a(o_{t-1}) < f_p^a(o_t)$. In Fig. 8.4, this would be the case for both skills a_{red} and a_{yellow}, if the robot drives backwards whereby it moves away from the goals.

These heuristics lead to the voting function in Eq. (8.18).

$$P_a(o_t \,|\, o_{t-1}) = \begin{cases} \min\left(\max\left(\frac{f_p^a(o_t) - f_p^a(o_{t-1})}{1 - f_p^a(o_t)}, 0\right), 1\right), & 1 - f_p^a(o_t) < \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{8.18}$$

This voting function is then used to accumulate the total votes for an observation over the period of one state transition. To discourage actions with long duration but low accumulated votes, it is divided in addition by the time span of the full state transition:

$$P_a(s_{t_2} \,|\, s_{t_1}) = \frac{\sum_{t=t_1}^{t_2} P_a(o_t \,|\, o_{t-1})}{t_2 - t_1} \tag{8.19}$$

For each observed potential state transition $\langle s_{t_1}, s_{t_2} \rangle$, the robot uses Eq. (8.19) to determine the most likely transition action

$$a_{ml} = \arg\max_a P_a(s_{t_2} \,|\, s_{t_1}) \,. \tag{8.20}$$

This action determines the transition probability in the observer's strategy that would most probably correspond to the observation of the demonstrator:

$$P(s_{t_2} \,|\, s_{t_1}) = T(s_{t_1}, a_{ml}, s_{t_2}) \tag{8.21}$$

Integrating Eq. (8.16)-(8.21) into Eq. (8.11) leads to the following modified recursive solution:

$$\mathcal{V}(s,t) = P(o_t \,|\, s_t = s) \max_{s'} \left[T(s_{t-1} = s', \arg\max_a P_a(s_{t_2} \,|\, s_{t_1}), s_t = s) \mathcal{V}(s', t-1) \right] \tag{8.22}$$

In contrast to the original Viterbi approach, no initial state probabilities can be assumed for the demonstrator. Hence, $\mathcal{V}(s, 1)$ in Eq. (8.12) is being simplified to

$$\mathcal{V}(s,1) = P(o_1 \,|\, s_1 = s) \;\forall\; s \in \mathbb{S} \,. \tag{8.23}$$

$\varphi(s,t)$ is finally determined according to Eq. (8.13). As the observed actions are not of unit time, the imitator will recognize the same action and state for some period of time. The recognized state-action transitions must therefore be extracted from φ to be of any use to the imitating robot. An example is given in Eq. (8.24), where the tuples show the condensed information extracted from φ.

$$\overbrace{s_0 \to s_0 \to s_0 \to s_2}^{(s_0,a_1,s_2)} \to s_2 \to \overbrace{s_1 \to s_1 \to s_1 \to s_3}^{(s_1,a_3,s_3)} \qquad \underbrace{s_2 \to s_2 \to s_1}_{(s_2,a_2,s_1)} \tag{8.24}$$

While observing the demonstrator, it can happen from time to time that the imitator does not find corresponding actions for the observations, i. e., $P_a(o_{t^o_{obs}} \,|\, o_{t^o_{obs}-1}) < \theta_{obs}$ for all actions. This

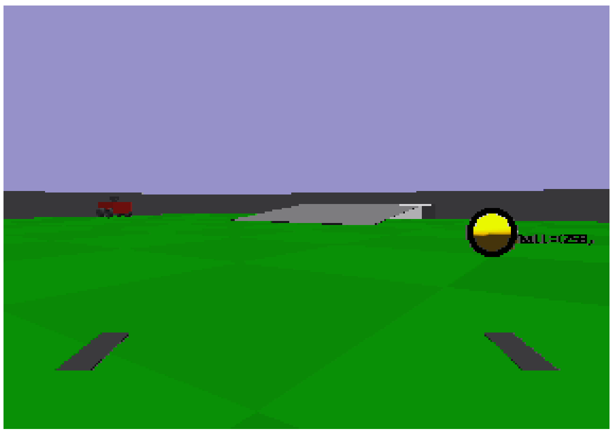


Figure 8.5: The subjective perception of the robot used in the experiments. It shows one camera image with the ball that has been recognized by the vision preprocessing step.

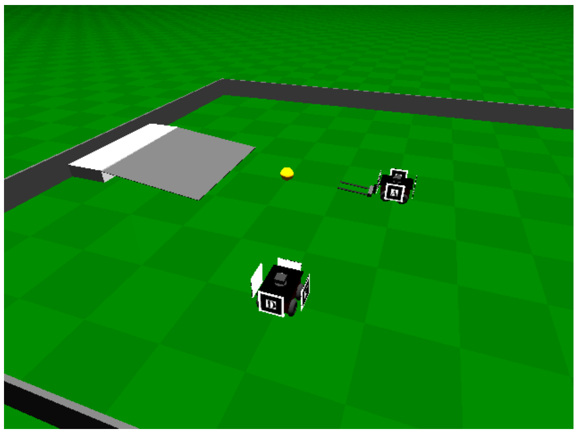


Figure 8.6: Experimental scenario: the ball has to be put onto the elevated platform.

means that the just observed behavior at time t^0_{obs} is unknown to the imitator. The original Viterbi algorithm would zero out all subsequent probability values, which would result in no sensible recognition at all: $\mathcal{V}(s,t) \approx 0 \; \forall t > t^0_{obs}$. To prevent this, the demonstrator suspends the normal Viterbi path calculation and scans the observation stream for the next time step t^1_{obs}, at which the demonstrator is again able to recognize a skill with sufficient confidence: $P_a(o_{t^1_{obs}} | o_{t^1_{obs}-1}) \geq \theta_{obs}$. The threshold θ_{obs} has to be determined empirically. The algorithm then starts the described modified Viterbi algorithm anew at time t^1_{obs}. It returns these independently recognized episodes as separate traces:

$$\Gamma = (\ldots, ((t,o,s), a, (t',o',s')), \ldots) \tag{8.25}$$

For full reference, the whole algorithm is depicted in Alg. 2 and 3 (pages 141 and 142).

8.4.3 Example

To demonstrate the recognition process, two robots are placed into an environment with a soccer ball that has to be transported onto an elevated platform (Fig. 8.6). To achieve this, they can either simply push it or use their grippers to pick the ball and release it onto that platform. In this example, both robots have fixed roles, subsequently called demonstrator and imitator, and the imitation period is given to focus on the recognition part.

The robots have a defined field of view (fov) of 60°. They are able to perceive the soccer ball's position and width in the camera image if it is within their fov by means of their vision capabilities. The platform onto which the ball has to be placed is given as absolute coordinates to the robot, which also knows its own position. The robots are equipped with 2-D barcode markers, which enable them to detect each other's relative position and orientation by means of the camera based tracking library ARToolkit [183]. The raw perception $\mathcal{I}$ provides the following information to the imitator (Fig. 8.5):

$$p = \begin{pmatrix} \boldsymbol{v}_b^I \\ \boldsymbol{v}_g^I \\ (\boldsymbol{v}_D^I, \boldsymbol{\theta}_D^I) \end{pmatrix} \begin{array}{l} \text{ball's relative 3-D coordinates} \\ \text{goal platform's relative 3-D coordinates} \\ \text{3-D vector to and bearing of the demonstrator robot} \end{array}$$

For the strategy layer, the raw perception $\mathcal{I}$ is then converted into a raw state observation $o = \mathcal{I}_s$ by the user-defined function Υ_s (cf. Sec. 8.3 and Chap. 3):

$$o^I = \Upsilon_s(\mathcal{I}^I) = \begin{pmatrix} \|\boldsymbol{v}_b^I\| \\ \|\boldsymbol{v}_g^I\| \\ \boldsymbol{v}_g^I \cdot (0,0,1) \end{pmatrix} \begin{array}{l} \text{ball distance} \\ \text{goal distance} \\ \text{ball height} \end{array} \tag{8.26}$$

As described in the previous section, the recognition algorithm does not operate on its own state observation o, but on the observation transformed into the demonstrator's view:

$$o^D = \Upsilon_s(\mathcal{I}^D) = \begin{pmatrix} \|\varphi_D(\boldsymbol{v}_b^I)\| \\ \|\varphi_D(\boldsymbol{v}_g^I)\| \\ \|\varphi_D(\boldsymbol{v}_b^I \cdot (0,0,1))\| \end{pmatrix} = \begin{pmatrix} \|\boldsymbol{v}_b^D\| \\ \|\boldsymbol{v}_g^D\| \\ z_b^D \end{pmatrix} \tag{8.27}$$

The skill layer is equipped with the capabilities to rotate and translate the robot and to use the gripper. If it is not using its grippers, it is nevertheless able to move the ball around in the field by simply pushing them. In the imitation process, only the positions of the ball and of the other robot can be observed.

The robots are capable of appropriate strategies and skills in order to move the ball around. The demonstrator disposes of three skills: approaching the ball, lifting the ball, and approaching the goal. The imitator is lacking the skill to lift the ball. It has instead individually learned the skill to approach the ball (cf. Chap. 6) and is able to approach the goal and the corresponding strategy using those skills (cf. Chap. 5). In the experiment, the imitator is observing the demonstrator how it moves to the ball, picks it up, and carries it to the platform. It then tries to recognize a potentially beneficial strategy in the observation.

The outcome of the recognition process is shown in Fig. 8.7. The imitator has successfully recognized episodes in the demonstrator's movements that coincide with the imitator's own behavior knowledge (dark areas marked as "B" and "G"). *B* denotes the time span in which the demonstrator recognized a skill resembling its own *approach ball* skill and *G* resembling its *approach goal* skill. The time span between the recognized areas is detected by the imitator as *not understandable* behavior (light areas). The recognition process is bootstrapped again as soon as it has reasonable explanations for the observed behavior data. This *missing link* can now be used in the subsequent exploration processes to direct the exploration towards it, while the understandable regions can be used, e. g., to adapt the strategy towards using them more aggressively.

If the imitator would have been able to lift the ball, the corresponding recognition results would be as depicted in Fig. 8.8. The time span that was missing in Fig. 8.7 is now correctly detected as the behavior *lifting the ball* and marked with "L".

In this section, the focus has been on the recognition part. In the following, it will be used in a larger picture, when recognized behavior is integrated in the robots' own knowledge.

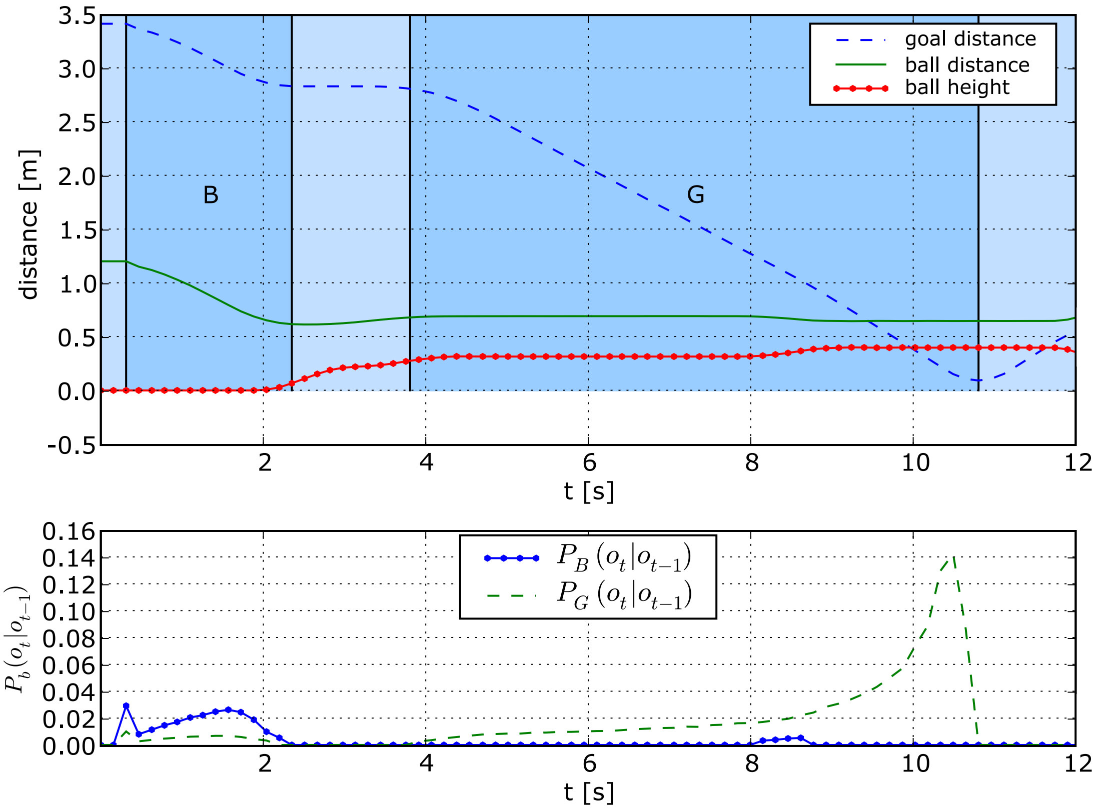


Figure 8.7: Recognition results during the imitation process: B and G (dark areas) denote the behavior over the time that the demonstrator has understood and interpreted as being similar to its own *approaching ball* and *approaching goal* skills. The behavior between them, lifting the ball (light area), is recognized as a missing link.

8.5 Integrating recognized behavior

Although the previously described recognition algorithm is the most important part in the imitation process, more efforts have to be made for a successful imitation process. The most natural way is to integrate them in the form of interactions to the strategy's own experience so that the strategy layer does not have to discriminate between own and observed experiences.

Recall from Sec. 8.4 that the recognition algorithm (Alg. 2) results in sequence of understood state transitions

$$\Gamma = (\ldots, ((t, o, s), a, (t', o', s')), \ldots) \ . \tag{8.28}$$

The imitation data has to be delivered to the strategy in the form of interactions, according to the strategy's experience stream by Eq. (5.1) (page 33):

$$I_{t_1}^{t_2} = (o_{t_1}, a_{t_1}, d_{t_1}, \boldsymbol{\mu}_{t_1}, f_{t_1}, o_{t_2})$$

The missing data duration d, motivation $\boldsymbol{\mu}$, and failure f need to be reconstructed for the returned data of the recognition algorithm. Obviously, the duration is the difference of the start and ending of the observation.

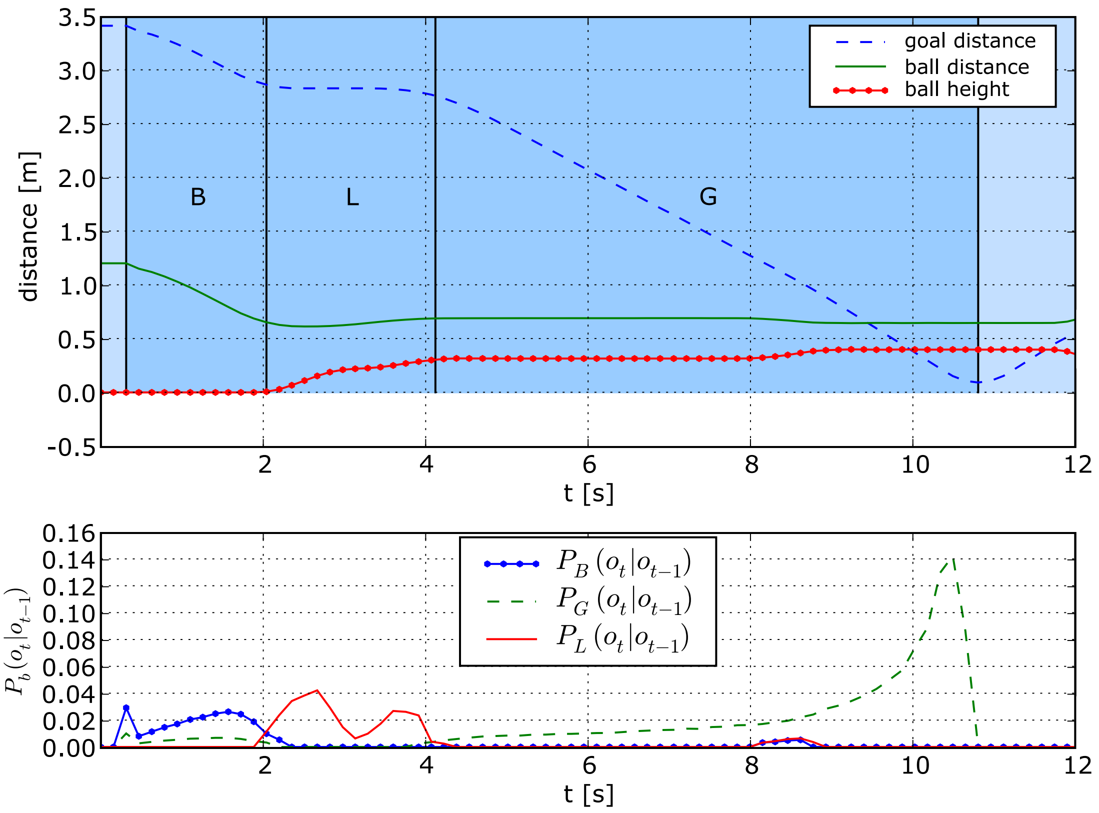


Figure 8.8: Recognition results during the imitation process when the robot is capable of lifting the ball: instead of the missing link the robot has correctly detected the lifting skill.

The motivation $\boldsymbol{\mu}$ is not directly observable. A way to infer the outcome of the recently performed behavior is nevertheless vital for successful imitation. For this reason, the demonstrator is required to express additional information, which is sufficient to approximate its overall drive state $\boldsymbol{\mu}$. In the following experiments, a colored light bulb on top of each robot achieves this. The imitator approximates the motivation $\hat{\mu}$ for each observed state transition from the drive state expressed by the demonstrator at time t. Practically, this is done by normalizing the observed drive state expression μ^D with respect to its minimum $\mu^D_{\min}$ and maximum $\mu^D_{\max}$ for each demonstrator robot d individually, including the imitator itself. This requires an initial calibration phase where rough estimates are determined and only observation is allowed. Afterwards, the imitator processes the observation using the approximated reward $\hat{\mu}$:

$$\hat{\mu} = \mu^D \cdot \frac{\mu^I_{\max} - \mu^I_{\min}}{\mu^D_{\max} - \mu^D_{\min}} \tag{8.29}$$

The demonstrator keeps monitoring the minima and maxima for a better reward estimation.

The failure signals f are all assumed to be false. With a, d, and $\boldsymbol{\mu}$, the robot is then able to create a trace of interactions, containing only the most salient information ($|I^{t_q}_{t_1}| \ll |O^{t_N}_{t_1}|$):

$$I^{t_q}_{t_1} = \langle I_{t_1}, I_{t_2}, \ldots, I_{t_q} \rangle \tag{8.30}$$

Based on $I^{t_q}_{t_1}$ the imitator has to decide whether it is worth to integrate it into its own experience

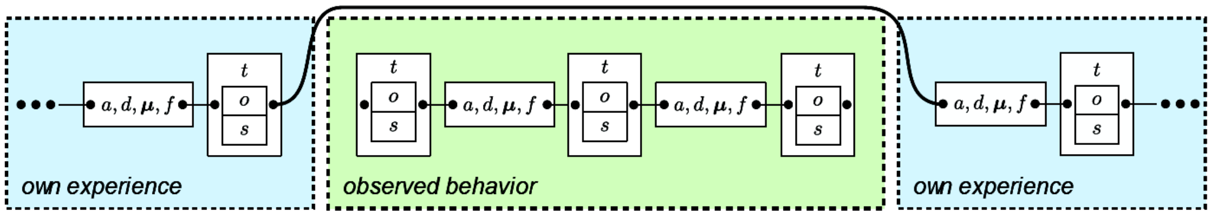


Figure 8.9: The experience in the form of an interaction stream. The recognized behavior, which is observed at other robots, is inserted into the experience as a new stream. Afterwards, the stream of the own experience continues. This makes it possible to acquire observed data, while actively collecting further experience.

at all. The demonstrator might have been in exploration phase and thus have chosen random actions. Alternatively, it might have been in exploitation phase, but its behavior does not convey much vital information. In both cases, the approximated rewards are a valid indicator for the usefulness of $I_{t_1}^{t_q}$. If the demonstrator experienced high absolute rewards, it does not matter whether it was in exploration phase while being observed by the imitator. On the other hand, if the demonstrator did not show any significant change in its overall well-being, it is irrelevant whether it was in exploitation mode.

If the imitator decides that it can benefit from the interaction trace, it appends $I_{t_1}^{t_q}$ to its strategy's experience. However, it has to be implemented as a new episode of experience – not connected to the robot's own stream of experience, as shown in Fig. 8.9. Otherwise, the state transitions would be inconsistent. The robot backs up its current state and restores it again after the imported experience to prevent having states of two different actors in one interaction. Otherwise, the inserted interaction would look like a teleportation through the world. The strategy layer keeps track of all "dangling" episodes in the experience stream and incorporates them accordingly when updating its policy. In this manner, the whole process of learning observed behavior is transparent to the underlying strategy. It does not know whether its input originates from an observation or from its own perception. Thereby, the imitator benefits from the observation utilizing the complete strategy learning.

8.6 Evaluation

The overall imitation approach is evaluated in two Capture-The-Flag (CTF) scenarios, where the robots had to transport objects to goals of different value. To evaluate the benefits of the imitation approach, all but one goal are easily approachable and provide only a low reward, while one goal is difficult to reach, but provides a high reward. The optimal strategy, carrying all objects to the difficult but highly rewarding goal, is learnable both with and without imitation. The question is whether imitation results in a faster learning process.

The simulation models of the well-known Pioneer2DX is extended with an LED, which expresses the corresponding well-being states (Fig. 8.10). The perception is preprocessed and delivered to the strategy layer as a three-dimensional vector containing the following information:

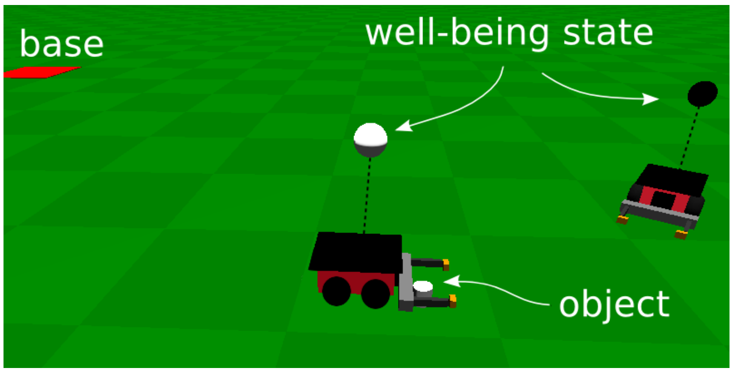


Figure 8.10: The robots have grippers to grab the objects and LEDs for showing their emotional state

- Distance of robot to the closest object,
- Distance of the closest object to the closest goal, and
- ID of the closest goal

All experiments were conducted twice: with the imitation activated and with the imitation deactivated. Wherever appropriate, the charts contain a 95% confidence-interval.

The development of the group's behavior homogeneity is analyzed using Shannon's information entropy [161], which is a measure for the disorder of the realizations of a random variable:

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{8.31}$$

H's range is $[0, H_{max} = \log |X|]$, so it is normalized and the resulting function G is used as a measure for the emergence of order in a strategy:

$$G(X) = \frac{H_{max} - H(X)}{H_{max}} \tag{8.32}$$

In the experiments, X represents the goals to which the objects are transported. With $G(X) = 0$ all goals are chosen equally often. As $G(X)$ is approaching 1, the robots prefer more and more one goal over the others. By comparing the normalized entropy of the imitation to the no-imitation case, it can be seen whether imitation also has sped up the convergence of the group behavior.

8.6.1 CTF with three bases

This scenario consists of the three goal bases *red*, *yellow*, and *black*, to which the objects, which are dispersed in the field, have to be transported (Fig. 8.11). The robots have predefined skills, which are provided by the skill layer described in Chap. 6: approach the nearest object, approach the red base, approach the yellow base, and approach the black base. The skills remain fixed during the complete experiment. A positive reward of 10 is given for collecting an object. Once the object has reached one of those bases, it receives an additional positive reward. For the yellow

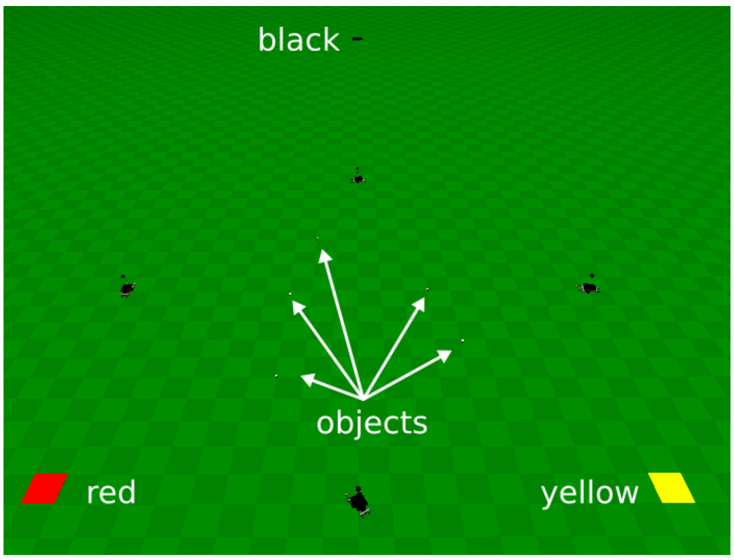


Figure 8.11: Scenario with three bases. For delivering the objects to the red or yellow base at the bottom, the robots receive a reward of 20. For the black base the reward is 10,000.

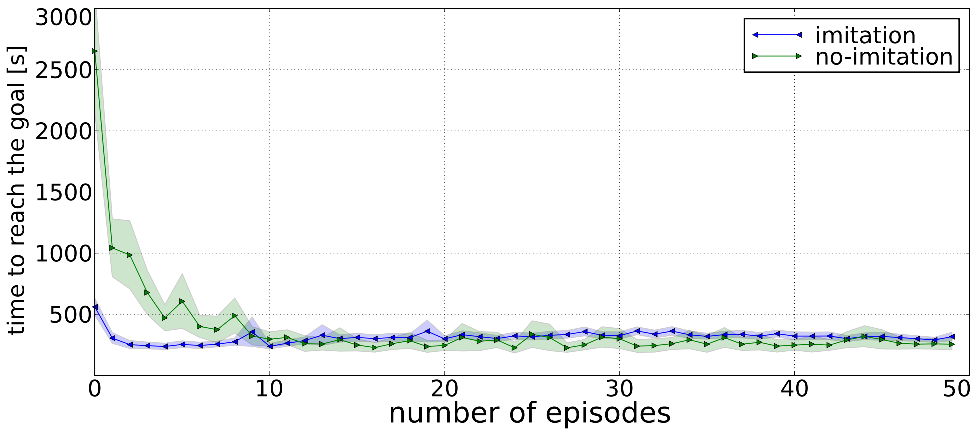


Figure 8.12: The average time needed for reaching a goal over 50 runs

and red base the reward is 20 points. The black base is farther away and thus more difficult and unlikely to reach. For transporting an object to this base, a robot receives 10, 000 points. The values for all charts are averaged over more than 500 times that a robot transports an object to a goal.

The time a robot needed to catch one object and deliver it to a base is shown over 50 consecutive episodes in Fig. 8.12. Evidently, the experiment with the imitation activated is much faster than the one with the imitation deactivated in the beginning. The curves meet each other after eight episodes and stay nearly the same, with a slight advantage for no-imitation. The reason for the no-imitation version being a little faster in the end does not indicate that it is better. This is the case, because the robots in the no-imitation experiment visit the black base less frequently. As the black base is much farther away, it naturally leads to a shorter average time to reach the goal. With activated imitation, all goal bases nearly get the same amount of objects in the beginning.

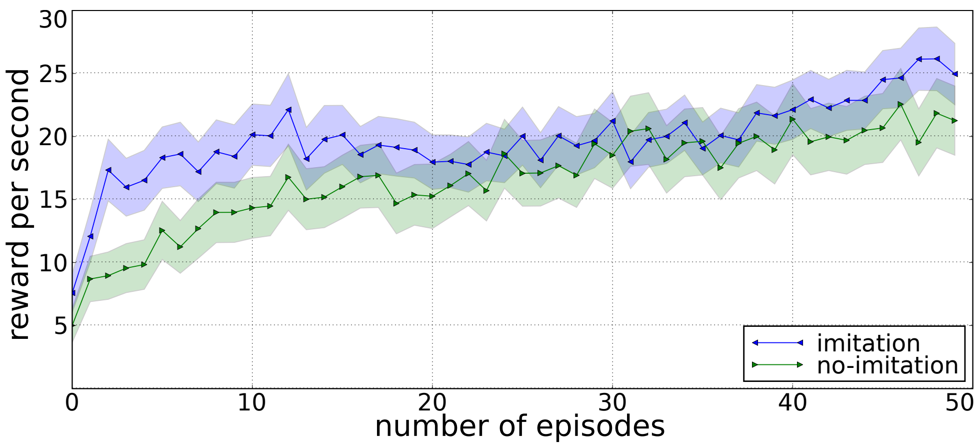


Figure 8.13: Reward per second

After some exploration, increasingly more robots find out that it is beneficial to prefer the black base (Fig 8.15). Without imitation the robots explore and learn to prefer the black base, too (Fig. 8.14). However, the number of robots knowing this fact is much lower than the number of robots with the activated imitation. Therefore, the average time to reach a goal is higher for activated imitation, because the distance to the black base is farther.

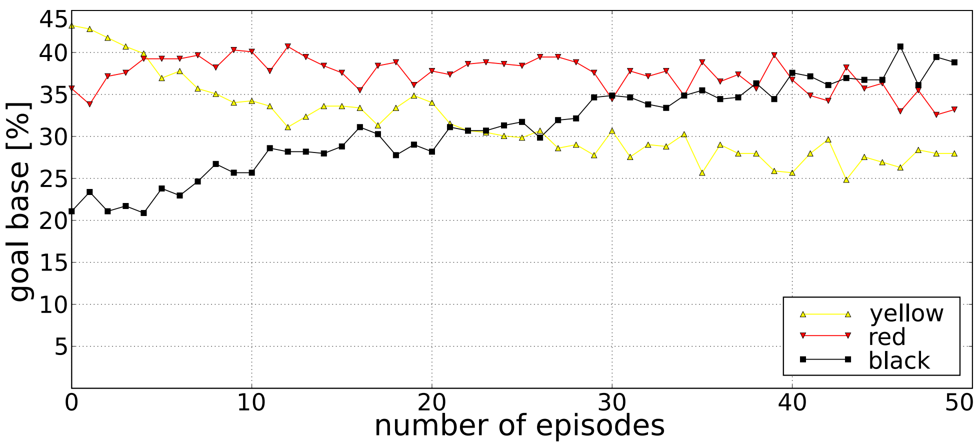


Figure 8.14: The percentage of objects brought to the respective base *without* imitation

The reward per second in Fig. 8.13 is a good overall indicator of how successful a strategy is. It takes into consideration the reward as well as the time needed to receive it. Imitation starts better than no-imitation because of the shorter amount of time needed to reach the goal. Later on, both values are similar. At the end, the required time does not drop anymore, but the average reward increases with more robots choosing the black base. As can bee seen in the chart the imitating robots are in advantage. Due to the imitation process, the robots are able to faster learn the more rewarding but also more time consuming behavior than with just individual learning.

To get a better understanding, Fig. 8.16 shows the amount of experiences in terms of interactions the robots have made in each episode. It is interesting that the imitation starts with a lower amount of experiences, although the only difference between both versions is that imitation ac-

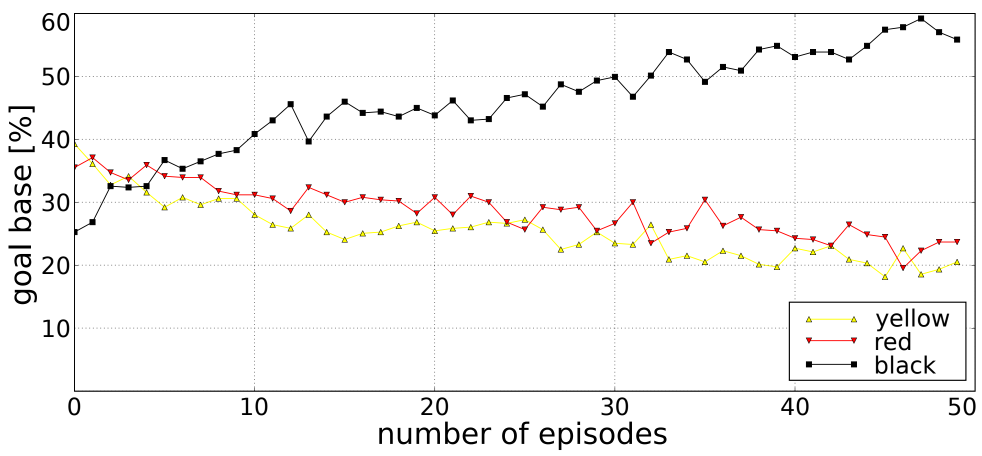


Figure 8.15: The percentage of objects brought to the respective base *with* imitation

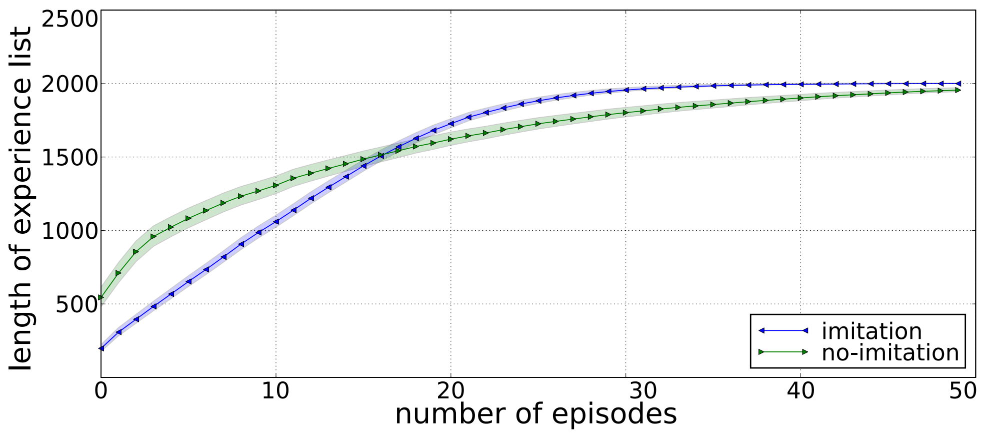


Figure 8.16: Size of the experience list

quires more information by means of observation. The acceleration achieved by imitation is so high that the necessary amount of experiences to reach a goal is much lower than without imitation. After some time, both charts cross each other, because the time, which is needed to reach a goal, becomes nearly the same in both cases. At the end, imitation has acquired more experiences by observation. The number of experiences is bounded by 2,000, where old observations are dropped as new ones arrive. Using this sliding experience horizon, the robots are able to adapt to changing environments and prevent information drowning.

Another question is how well the whole approach managed to handle the additional complexity, introduced by the imitation process. This can be seen in Fig. 8.17. It shows the number of abstract regions, into which the heuristics have divided the state space (cf. Sec. 5.5). It shows that the graph converges below ten regions for imitation and even below six for no-imitation. The layered learning architecture of the robots is able cope with the complexity of the task by maintaining an appropriate state space, which consists of a limited number of abstract regions.

Finally, the group behavior is analyzed with respect to emerging behavior patterns. Emergence is the way patterns in complex systems arise out of a multiplicity of relatively simple interactions.

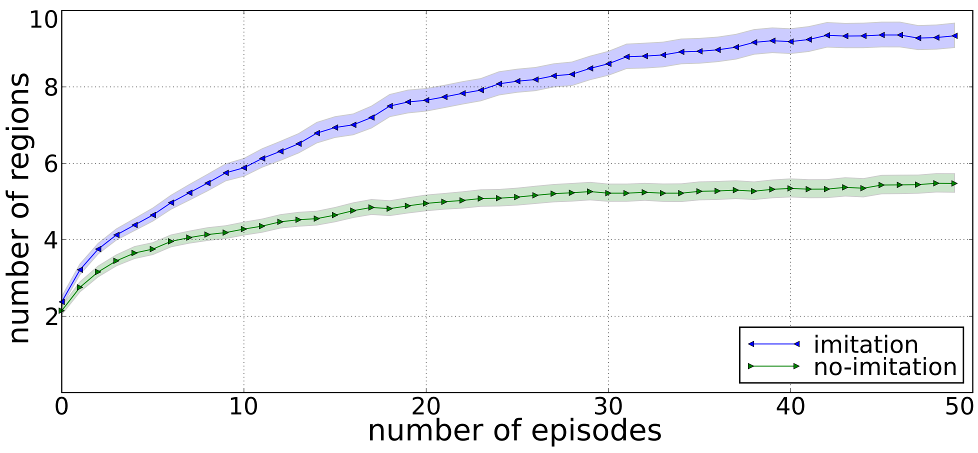


Figure 8.17: Number of regions, in which the state space is split

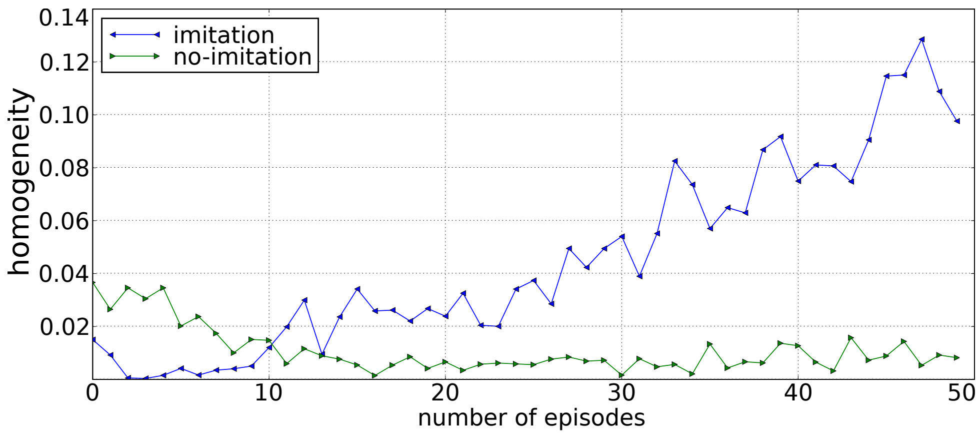


Figure 8.18: Goal choice homogeneity (1 = goals are maximal homogeneous)

Fig. 8.18 displays the emergence of the chosen goal base in each episode. A value of zero means that the chosen goals are maximally heterogeneous. The value for maximum homogeneity is one. With imitation enabled the emergence increases significantly while no-imitation is stuck at a low value. So, the class of imitating robots shows much more homogeneity in their chosen goals than the class of individual robots that do not learn from each other.

8.6.2 CTF with five bases

In this scenario two additional goal bases were added, as shown in Fig. 8.19. Both bases were placed between the black base and the other low-value bases. Reaching an object did not result in additional reward. All other parameters stayed the same as in Sec. 8.6.1. With the additional two actions necessary to reach the two new goals the whole scenario gets more complex. The fact that the new bases are between the objects and the black goal base decreases the chance that a robot ever drives to the black base.

Fig. 8.20 and Fig. 8.21 show the distribution of the objects carried to the different goal bases.

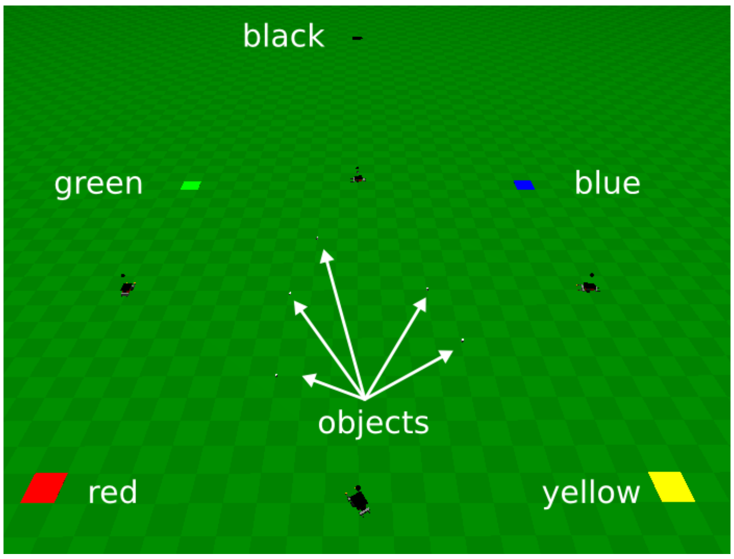


Figure 8.19: Scenario with five bases: for delivering the objects to the blue, green, red, or yellow base, the robots receive a reward of 20. The reward for the black base is 10, 000.

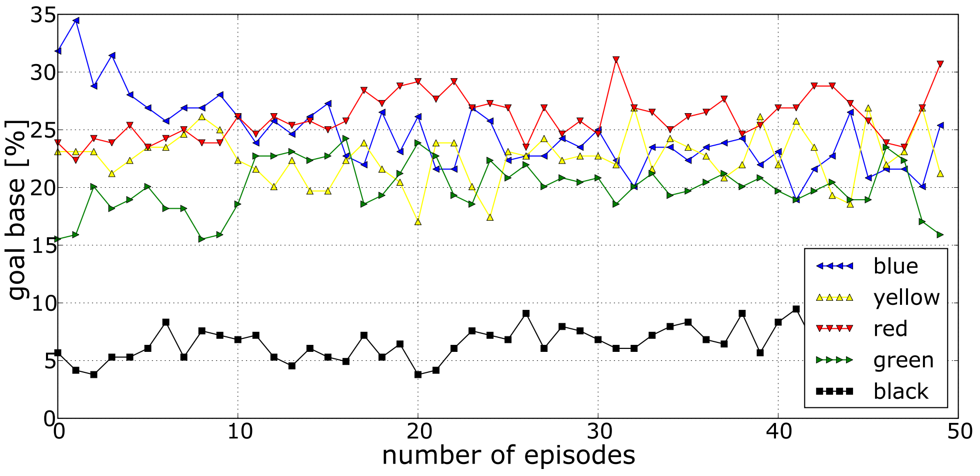


Figure 8.20: The percentage of objects brought to the respective base *without* imitation

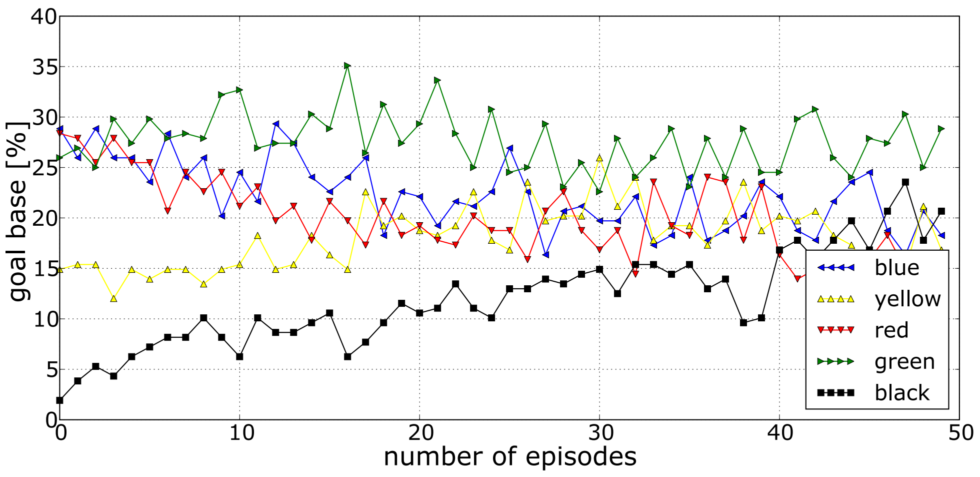


Figure 8.21: The percentage of objects brought to the respective base *with* imitation

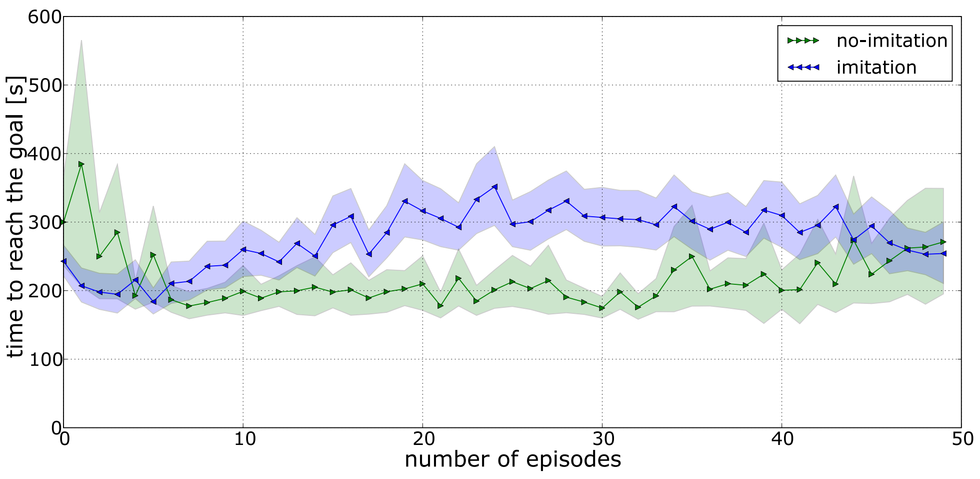


Figure 8.22: The average time needed for reaching a goal over 50 runs

The black base gets less objects compared to the previous experiments. Still, imitation shows a significant improvement of the black base, starting from 2% up to 20% while the no-imitation version never exceeds 10%.

This is also being underlined in Fig. 8.23, showing the reward per second, which the robots receive. Initially, the imitation version shows similar performance as the learning-only version. With time, imitation gets better than no-imitation, though.

The five base experiment points out that also in the more complex scenario robots that imitate are at an advantage. Having two more goal bases, the scenario decreases the probability to reach the black goal base. Without imitation, the goal base distribution stays nearly the same. With imitation enabled, it increases considerably. Imitation improves the overall performance by spreading the information about beneficial behavior faster in the group.

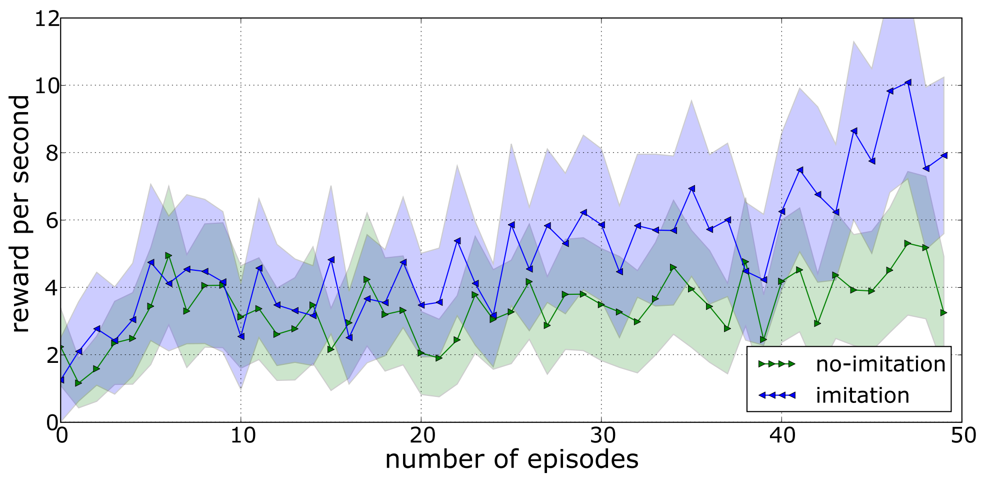


Figure 8.23: Reward per second

8.7 Conclusion

This chapter has shown how imitation improves the learning speed and performance of a robot group. The presented approach did so by finding the maximum likely path of states that corresponds to the observation with full reference to the imitator's own knowledge. With it, the imitator could reliably explain the demonstrator's behavior in terms of its own capabilities.

The imitating robot only used data, which is externally perceivable, and did not require the demonstrating robot to reveal its internal states or actions. Thereby, imitation is not restricted to robots anymore that have explicitly been prepared for that beforehand. Using the presented approach robots can now improve their strategies observing any robot that is around only requiring that it expresses its overall state. This greatly enhances the autonomy of future multi-robot systems.

CHAPTER 9

Choice of the imitatee

So far, it has been assumed that the behavioral capabilities of the robots in the group are similar. This means that all the robots are morphologically homogeneous and have the same algorithmic capabilities. In this case, it does not matter whom a robot imitates, because all robots are assumed to be equal. Furthermore, they should observe one another all the time. This is, however, not the case any longer in heterogeneous robot groups. Instead, prior to the imitation act itself, a robot intending to imitate first has to choose a robot, whose demonstrated behavior is likely to result in a behavior improvement for the imitator. For this purpose, the robot needs a notion of similarity measurement for behavioral capabilities. How this may be realized and used to choose the best imitatee[1] in a group of heterogeneous robots will be presented in this chapter [1][2].

In summary, the approach constructs and maintains at runtime for each robot in the group an *affordance network*, which is a Bayesian network of *affordances* detected in the observation. These networks encode dependencies about the interaction possibilities, which are offered by environmental objects to the different robots. Using the concept of affordances, a robot is able to reason about behavioral differences between robots without having to take into account their diverse hardware and software conditions. With a metric on those networks, a robot is able to calculate the difference between its own capabilities and those of the other robots. Prior to its imitation process, the robot can then choose the robot in the group that has the smallest behavioral distance to itself in order to maximize the probability of the imitation success.

[1]Throughout this chapter "imitatee" and "demonstrator" will be used interchangeably.
[2]This approach has been implemented within the scope of the diploma thesis [82].

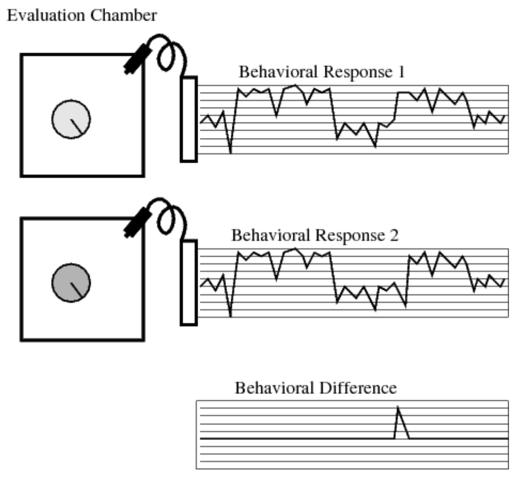


Figure 9.1: Measurement of behavioral difference according to Balch [32] in an idealistic evaluation environment, which is represented by the box with the circle representing the robot. For every possible situation the actions of the two robots under investigation are recorded and plotted at the right-hand side. Summation of the actions' differences is defined as the behavioral difference (bottom)

9.1 Related work

Attempts in this regard have already been made by Balch [33]. He devised an approach to calculate the *Hierarchic Social Entropy* of robot groups, which is a modification of Shannon's *Information Entropy* [161]. One step in his approach is the calculation of the behavioral difference between two robots. Therefore, the difference in the chosen actions given the same state is summed over all possible states the robots might encounter (Fig. 9.1). Formally, given a group of robots $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, such that each robot $\mathcal{R}_j$ can choose an action a^i_j in state s_i of a discrete state space according to its policy $\pi_j : s_i \mapsto a^i_j$. With p^i_j being the fraction of time steps relative to its whole life span that robot $\mathcal{R}_j$ has spent in state s_i, the behavioral difference between robot $\mathcal{R}_a$ and $\mathcal{R}_b$ as defined by Balch is calculated by $D_B(\mathcal{R}_a, \mathcal{R}_b)$:

$$D_B(\mathcal{R}_a, \mathcal{R}_b) = \sum_i \frac{(p^i_a + p^i_b)}{2} |\pi_a(s_i) - \pi_b(s_i)|. \tag{9.1}$$

In the case that $\mathcal{R}_a$ and $\mathcal{R}_b$ choose the same action in each state, $D_B(\mathcal{R}_a, \mathcal{R}_b) = 0$. If, in the opposite, they disagree all the time about the best action to choose, $D_B(\mathcal{R}_a, \mathcal{R}_b) = 1$.

Although this approach of calculating the behavioral difference between two robots could theoretically be used to determine the most similar demonstrator for an imitator, it is hardly applicable outside of laboratory environments: p^i, π, and s_i are required for all possible states both robots have encountered. Even if that would have been possible, the approach is restricted to a robot group, in which the same performed action $\pi_a(s_i) = \pi_b(s_i)$ leads to the same effects in the environment for both robots $\mathcal{R}_a$ and $\mathcal{R}_b$. The robot group must be totally homogeneous not allowing for the slightest manufacturing tolerance and requiring the same action set for all robots.

Shen et al. developed means to detect the similarity and synchronicity between the behavior of a human and a robot [162]. They detect spatial and temporal relationships between events in the perception stream of the robot using Crutchfield's information distance [63]. It measures the distance between two information sources and is based on Shannon's information entropy [161]. The robot analyzes the trajectories of vision-based markers (ARToolkit [183]), which are attached to the body parts of a human. In experiments with a humanoid robot, the approach manages to detect similar behavior like arm waving even if the behavior is time shifted. When transferred to multi-robot scenarios, the approach of Shen et al. is restricted to application settings where all participating robots share a similar morphology. Furthermore, it requires a fixed morphological mapping between all the robots in the group. In essence, this requires to manually resolve the correspondence problem (cf. Sec. 2.2.2.1).

In this chapter, a demonstrator selection approach is presented, which solves this challenge in an unobtrusive manner. Only relying on observable information that can be subjectively perceived, it helps a robot to find the robot in a robot group that is most similar to itself. Unlike the behavioral difference approach of Balch, this approach does not need any access to the internal states of the observed robots' high-level strategy or data structures of their low-level behavior. Nor does the approach assume the same action set for all participating robots in the group. In contrast to the approach of Shen et al., the approach presented in this chapter does not require the correspondence problem to be solved manually beforehand.

9.2 Background

Before presenting the demonstrator selection approach, this section will provide the basics behind its main ingredients: learning Bayesian networks and the nature of affordances.

9.2.1 Bayesian networks and how to learn them

A Bayesian network embeds dependency relationships between random variables [131]. The main purpose of a Bayesian network is to allow reasoning under uncertainty.

Definition 9.1 (Bayesian network) *A Bayesian network (BN) $B = (G, \Theta)$ on a set of random variables $X = \{X_1, \cdots, X_n\}$ is defined by two components:*

1. *A directed acyclic graph $G = (X, E)$ with nodes representing the random variables and edges $E \subseteq X \times X$ encoding the conditional dependencies between them. G is also called the* structure *of B.*
2. *A set $\Theta = \{\Theta_1, \ldots, \Theta_n\}$ of conditional probability tables (CPT) $\Theta_i = P(X_i \mid Pa(X_i))$, $i \in \{1, \cdots, n\}$, which are associated with the random variables.*
   - *$Pa(v) = \{u \mid u \in X, (u, v) \in E\}$ is the set of parent nodes.*

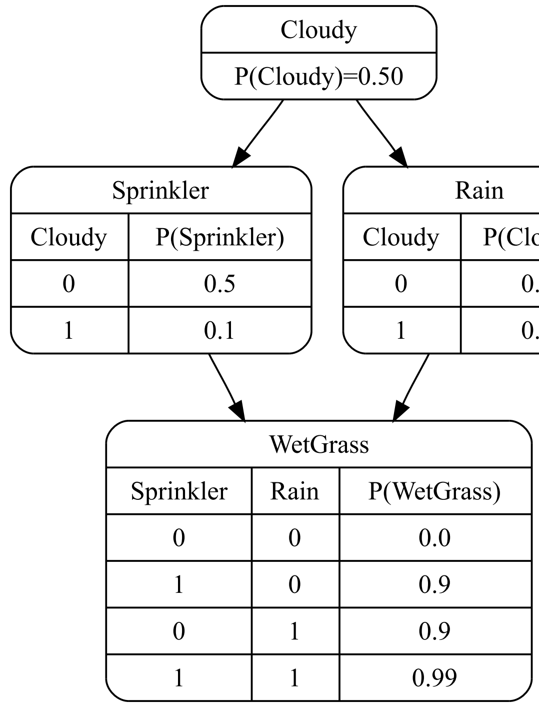


Figure 9.2: Exemplary Bayesian network [129]

- *The rows of the CPTs contain the probabilities* $\Theta_{ijk} = P(x_{ij} \mid ec_k(X_i))$ *of the* n_i *combinations* x_{ij}, $j \in \{1, \cdots, n\}$, *conditioned on the possible state combinations* $\boldsymbol{ec}(X_i)$ *of the parents* $Pa(X_i)$.
- $ec_k(X_i) \in \boldsymbol{ec}(X_i)$ *represents the k-th state combination of* $\boldsymbol{ec}(X_i)$. *The CPT of a node without any parents contains only unconditioned probabilities* $P(x_{i,j})$, *i. e.,* $\boldsymbol{\Theta_i} = P(X_i)$.

As an example, consider the situation where one sees wet grass outside and has to infer whether it is due to the sprinkler or because of the rain [129]. A BN with the conditional variables $Cloudy$, $Sprinkler$, $Rain$, and $WetGrass$ can be modeled based on past experience as a Bayesian network as shown in Fig. 9.2. It shows that the event that the grass is wet can have two possible reasons: either it has rained ($Rain$=1) or the sprinkler has been activated ($Sprinkler$=1). Each of those events again can have two possible events. For each combination, the Bayesian network shows the probability that the grass is wet. If, e. g., the sprinkler was off, but it has rained, the probability that the grass is wet is $P(WetGrass) = 0.9$. If one sees wet grass outside, using Bayes rule he can now determine whether it is more likely that the sprinkler was on or that it has rained. In this chapter, however, BNs are not used for inference, but for another reason. The conditional probabilities of a BN allow for a much more compact representation. The node $WetGrass$, e. g., is considered *conditionally independent* of the node $Cloudy$. The CPT of $WetGrass$ therefore does not include probabilities for $Cloudy$. As presented below, this will be advantageous, when BNs are used to encode behavioral dependencies of the robots.

If a Bayesian network BN is fully specified by its graph structure G and CPTs Θ, the probabilities of the random variables X_i may be calculated by summing the joint probabilities over all outcomes for their parents, which is called *marginalization*. In realistic applications, this is seldom the case, though. Often, some of the data and some of the BN's structure and/or CPT is given, and the remaining information for the BN has to be retrieved. This is also the case in the demonstrator selection approach. Here, the affordance data is given and the *most plausible* BN has to be found.

There are four different situations, in which BNs can be learned. The training data may be *complete* or *partly missing* and the network structure may be *known* or *unknown*. In this section, BNs will encode efficiently the dependencies of noisy, and sometimes missing, observations regarding the action capabilities of the surrounding robots. Thereby, the learning step faces the hardest situation: the data is incomplete and the network structure is not known in advance. This challenge is handled by the *Alternating Model Selection EM algorithm* by Friedman [76].

9.2.2 Affordances

As the choice of the demonstrator has to be performed prior to the imitation process itself, the robot has to be able to detect a set of behavioral capabilities in the other robots' performance. On the one hand, these behavioral capability detectors have to contain enough information to support an imitator in its choice of the demonstrator. On the other hand, they must be general enough to be able to detect the same pattern for morphological different robots.

A helpful concept for this was defined by the psychologist Gibson who observed that our perception of the world is dependent on our interactions with it [80]. He introduced the term of *affordance* defined as the action opportunity or interaction possibility the environmental objects present to an actor. An affordance is thereby a quality that an object offers to specific actors. Through the set of action possibilities, which an object offers to an actor, the object provides meaning to that actor, as seen by Overbeeke and Wensveen [137]:

> *The world appears to us as inherently meaningful because we perceive action possibilities, i. e., affordances. Meaning is in the world, directly, not inferred through reasoning.*

Thiruvengada and Rothrock analyzed affordances for their underlying properties. They found the following seven properties [177]:

1. It is an ecological concept defined at varying ecological levels for different animals.
2. An affordance is always attributed to a set of two or more things taken together.
3. By an affordance the environment informs the animal how it is to be used.
4. Sets of affordances describe a niche that specifies how a species lives.
5. An affordance contains real meaning, which exists independent of the perceiver.
6. Affordances are present even if a perceiver does not notice them.
7. An observer directly perceives basic affordances.

As such, affordances are a reasonable concept to detect robotic behavioral capabilities. When a robot is collecting information about the affordances that are offered to it by objects in the environment, it can compare whether its own niche resembles that one of another robot. The more similar the specific niches of two robots are the more similar the robots themselves can assumed to be and the more likely imitation will provide useful behavior.

The categorization of Zhang will help to define the type of affordance more clearly on which the demonstrator selection approach in this section bases on [190]. He distinguishes five different types of affordance:

Biological affordance is based on biological process. Some plants, e. g., afford nutrition, while others afford biological hazards.

Physical affordance is concerned with physical structures. A chair affords a human to sit on it, while it does not so to an elephant.

Perceptual affordance provides information cues regarding objects in the environment. Zangh offers the pictorial signs for ladies' and men's restrooms as examples.

Cognitive affordance are provided by cultural conventions, like traffic lights, e. g..

Mixed affordance combines several of the aforementioned affordances. A mailbox, e. g., provides no information to a person that has no knowledge (cognitive affordance) about its usages and structure of the mailbox itself (physical affordance).

By Zhang's definition, physical affordance is the appropriate type in this case, as the robots' capabilities are naturally found out by physical interactions with the objects. Physical affordances are reflected by the 3D structure of the objects involved in those affordances. Because 3D structures are not directly perceived by humans or animals, they must reconstruct them from the perceived 2D images. Since this seems to be an easy task for humans and animals, it is not so for state-of-the-art computer vision approaches [87, 73]. The concept of physical affordances is heavily used in current robotics research. Lörken and Hertzberg use it to ground planning operators of a mobile robot's planning task [116]. Stoytchev lets a robot manipulator arm learn affordances in experiments [170]. As another example, Detry et al. have figured out how a robot arm can learn object grasp affordances [66].

The purpose of the imitatee selection approach, however, is not the preprocession part of affordance recognition, but the use of affordances themselves in order to determine the best demonstrator for an imitation. Therefore, the physical affordances will be provided to the evaluation scenario. There, objects will offer affordances like *pushable*, *pullable*, *liftable*, or *seizable*.

9.3 Overview of the demonstrator choice process

The general overview of the process of choosing the best demonstrator for the later imitation process is outlined in Fig. 9.3. To support the explanation of this process, it will be described throughout this chapter subjectively from the view of an arbitrary robot $\mathcal{R}_m$.

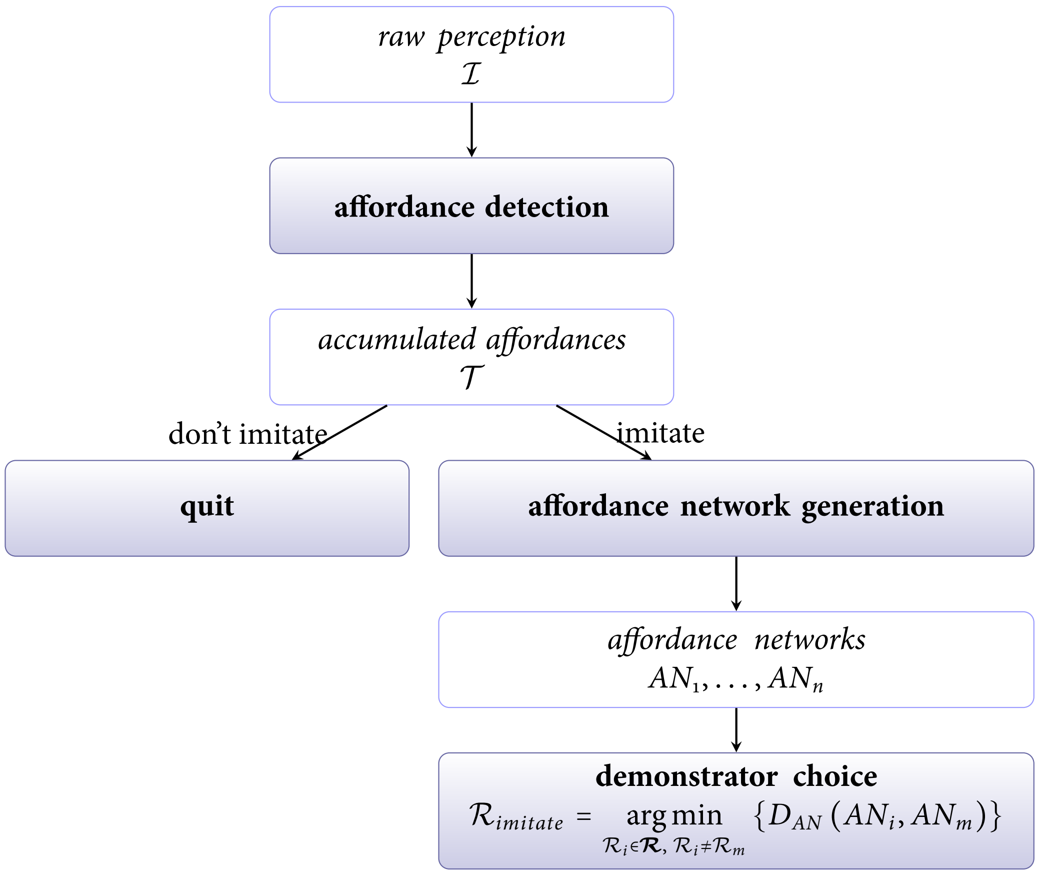

raw perception
$\mathcal{I}$
affordance detection
accumulated affordances
$\mathcal{T}$
don't imitate
imitate
quit
affordance network generation
affordance networks
$AN_1, \ldots, AN_n$
demonstrator choice
$\mathcal{R}_{imitate} = \underset{\mathcal{R}_i \in \boldsymbol{\mathcal{R}},\, \mathcal{R}_i \neq \mathcal{R}_m}{\arg\min} \{D_{AN}(AN_i, AN_m)\}$

Figure 9.3: Processes involved in choosing the best demonstrator for imitation

In order to acquire the necessary data, the $\mathcal{R}_m$ continuously monitors the other robots in the group and tries to detect affordances in the perception. These detected affordances are updated and accumulated in the affordance table $\mathcal{T}$. The process ends here if the robot is not about to imitate another robot.

If the robot plans to imitate another robot, it generates one affordance network (AN) for each robot in the group – including itself. These ANs are then compared to $\mathcal{R}_m$'s own AN. The best demonstrator is then determined as the robot who has the smallest distance to $\mathcal{R}_m$ in terms of their affordance networks.

9.4 Affordance detection

In order to measure the behavioral difference between two robots, the robot in question needs a sufficiently expressive set of measurable affordances [81]. These have to be provided beforehand. In specifying the detectable affordances of environmental objects, one has the possibility of specifying the complexity of the behavioral difference measurement. The more numerous and diverse the affordances are, which a robot is able to explore and observe, the more fine grained the robot can compare its own behavior to that of other robots. The advantage of using affordances is that they are completely independent of the robots morphologies.

In order to describe the affordance detection, let the set of robots in the environment be $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ and the set of object to be $\boldsymbol{O} = \{o_1, \ldots, o_q\}$. The perception stream $I_m(t) \in \mathcal{I}$ of robot $\mathcal{R}_m$ is assumed to be already preprocessed. In this perception stream, the robot is continuously looking for affordances $\Lambda = \{\Lambda_1, \ldots, \Lambda_p\}$ by means of one validity function for each affordance Λ_j:

$$Valid_j(I_m(t), o, \mathcal{R}_k) \in \{T, F, \perp\}\,. \tag{9.2}$$

It determines whether object $o \in \boldsymbol{O}$ offered robot $\mathcal{R}_k \in \boldsymbol{\mathcal{R}}$ the affordance Λ_j, with the robot $\mathcal{R}_m$ being able to observe itself. The validity function is composed of a list of conditions. All but the last condition are preconditions ensuring that the observed robot was able to test the affordance under question. If the test of one of these conditions fails, the affordance could not be determined, which is marked with an "$\perp$" in that case. Otherwise, the last check determines whether the affordance Λ_j is offered to the observed robot $\mathcal{R}$ ("T") or not ("F"). This distinction is necessary so that failed preconditions are not confused with lacking capabilities.

The following example will support the further explanations. The two robots $\mathcal{R}_{\text{red}}$ and $\mathcal{R}_{\text{blue}}$ are located in an environment with three objects from the set $\boldsymbol{O} = \{o_1, o_2, o_3\}$. As before, the data will be presented from the view of robot $\mathcal{R}_m = \mathcal{R}_{\text{red}}$. The filtered perception can be tested for the affordances $\Lambda = \{\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4\}$. After some time of observation, robot $\mathcal{R}_{\text{red}}$ will have collected the knowledge about which robot was offered what affordance by which object.

The knowledge is accumulated in $\mathcal{T}^{\text{red}} = \mathcal{T}^{\text{red}}_{\text{red}} \cup \mathcal{T}^{\text{red}}_{\text{blue}}$ as shown in Tab. 9.1: $\mathcal{T}^{\text{red}}_{\text{blue}}$ represents the data of $\mathcal{R}_{\text{red}}$ observing $\mathcal{R}_{\text{blue}}$:

$$\mathcal{T}^{\text{red}}_{\text{blue}} = \left\{(\Lambda_j, o_k, Valid_j(I_{\text{red}}(t), o_k, \mathcal{R}_{\text{blue}}) \mid j = 1, \ldots, 4,\ k = 1, \ldots, 3\right\} \tag{9.3}$$

$\mathcal{T}^{\text{red}}_{\text{red}}$ represents the data of $\mathcal{R}_{\text{red}}$ observing itself. $\mathcal{T}^{\text{red}}_{\text{red}}$ and $\mathcal{T}^{\text{red}}_{\text{blue}}$ differ only in the fact that $\mathcal{R}_{\text{red}}$ has been offered affordance Λ_4 by object o_3, in contrast to $\mathcal{R}_{\text{blue}}$. Although the information in the two data sets $\mathcal{T}^{\text{red}}_{\text{red}}$ and $\mathcal{T}^{\text{red}}_{\text{blue}}$ suffices to compare the robots, the direct comparison is unfeasible because of the following problems:

1. Data can be unknown ("$\perp$") or even missing (no entry in $\mathcal{T}$).
2. The perception is noisy. Boolean data has therefore to be processed to account for that.
3. The number of comparisons increases with the amount of collected affordance data.

In the next section it is shown how these problems can be solved by encoding $\mathcal{T}$ in a Bayesian network.

9.5 Affordance network generation

As the affordance data is noisy, instead of using $Valid_j(I(t), o, \mathcal{R})$ directly, one has to use the corresponding probabilities. Therefore, each affordance $\Lambda_j \in \boldsymbol{\Lambda}$ has to be associated with a random variable $A_j \in \{T, F\}$. This is done inside the Alternating Model Selection EM algorithm (cf.

Table 9.1: Affordance information, as detected by robot $\mathcal{R}_{\text{red}}$. The robots $\mathcal{R}_{\text{red}}$ and $\mathcal{R}_{\text{blue}}$ only differ in affordance Λ_4 offered by object o_3 (marked gray).

$\mathcal{T}^{\text{red}}$					
$\mathcal{T}^{\text{red}}_{\text{red}}$			$\mathcal{T}^{\text{red}}_{\text{blue}}$		
Λ_j	o_k	$Valid_j(I_{\text{red}}(t), o_k, \mathcal{R}_{\text{red}})$	Λ_j	o_k	$Valid_j(I_{\text{red}}(t), o_k, \mathcal{R}_{\text{blue}})$
Λ_1	o_1	T	Λ_1	o_1	T
Λ_2	o_1	$\perp$	Λ_2	o_1	$\perp$
Λ_3	o_1	T	Λ_3	o_1	T
Λ_4	o_1	F	Λ_4	o_1	F
Λ_1	o_2	F	Λ_1	o_2	F
Λ_2	o_2	$\perp$	Λ_2	o_2	$\perp$
Λ_3	o_2	F	Λ_3	o_2	F
Λ_4	o_2	F	Λ_4	o_2	F
Λ_1	o_3	T	Λ_1	o_3	T
Λ_2	o_3	T	Λ_2	o_3	T
Λ_3	o_3	T	Λ_3	o_3	T
Λ_4	o_3	T	Λ_4	o_3	F

Sec. 9.2.1). It replaces each observed "$\perp$" by the most likely assignment of T or F according to its internal heuristics [76].

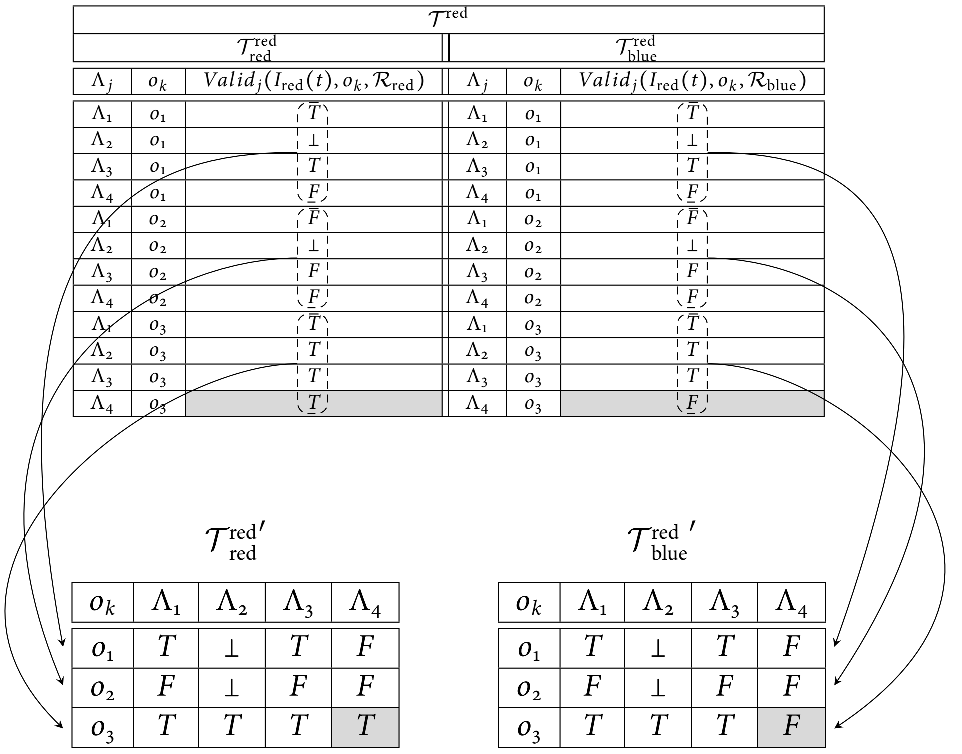
Robot $\mathcal{R}_m$ then approximates the unknown probability $P(A_j) = P(A_j = T)$, which represents the probability that affordance $\Lambda_j \in \Lambda$ is offered to robot $\mathcal{R}_l$:

$$P(A_j) = \frac{|\{o \mid Valid_j(I_m(t), o, \mathcal{R}_l) = T, o \in \boldsymbol{O}\}|}{|\boldsymbol{O}|} \quad . \tag{9.4}$$

Each triple $\left(\Lambda_j, o, Valid_j(I_m(t), o, \mathcal{R}_l)\right) \in \mathcal{T}^m_l$ is associated one element in the sample space of the random variable A_j. The data collected while interacting with its environment is now interpreted as one sample for each random variable A_j. The more data the robot collects the more accurate its approximation of $P(A_j)$ becomes.

Interpreting affordances as random variables has the following advantages:

- The behavioral capabilities of the robots are decoupled from the concrete objects as the comparisons can be made among the random variables in $\boldsymbol{A} = \{A_1, \ldots, A_{|\Lambda|}\}$ and not between the concrete triples in $\mathcal{T}^m_l$.
- The representation of the behavioral capabilities is more compact and robust as the uncertainties and incompleteness of the data is taken into account.
- The joint distribution of random variables nicely fits into the concept of Bayesian networks (BN) [131], which are very efficient and intuitive to interpret. The use of already existing graph metrics is possible as they are essentially graphs.
- By grouping affordances with respect to the objects, it is possible to reason over affordance dependencies.



$\mathcal{T}^{\text{red}}$

$\mathcal{T}^{\text{red}}_{\text{red}}$			$\mathcal{T}^{\text{red}}_{\text{blue}}$		
Λ_j	o_k	$Valid_j(I_{\text{red}}(t), o_k, \mathcal{R}_{\text{red}})$	Λ_j	o_k	$Valid_j(I_{\text{red}}(t), o_k, \mathcal{R}_{\text{blue}})$
Λ_1	o_1	T	Λ_1	o_1	T
Λ_2	o_1	$\perp$	Λ_2	o_1	$\perp$
Λ_3	o_1	T	Λ_3	o_1	T
Λ_4	o_1	F	Λ_4	o_1	F
Λ_1	o_2	F	Λ_1	o_2	F
Λ_2	o_2	$\perp$	Λ_2	o_2	$\perp$
Λ_3	o_2	F	Λ_3	o_2	F
Λ_4	o_2	F	Λ_4	o_2	F
Λ_1	o_3	T	Λ_1	o_3	T
Λ_2	o_3	T	Λ_2	o_3	T
Λ_3	o_3	T	Λ_3	o_3	T
Λ_4	o_3	T	Λ_4	o_3	F

$\mathcal{T}^{\text{red}\,\prime}_{\text{red}}$

o_k	Λ_1	Λ_2	Λ_3	Λ_4
o_1	T	$\perp$	T	F
o_2	F	$\perp$	F	F
o_3	T	T	T	T

$\mathcal{T}^{\text{red}\,\prime}_{\text{blue}}$

o_k	Λ_1	Λ_2	Λ_3	Λ_4
o_1	T	$\perp$	T	F
o_2	F	$\perp$	F	F
o_3	T	T	T	F

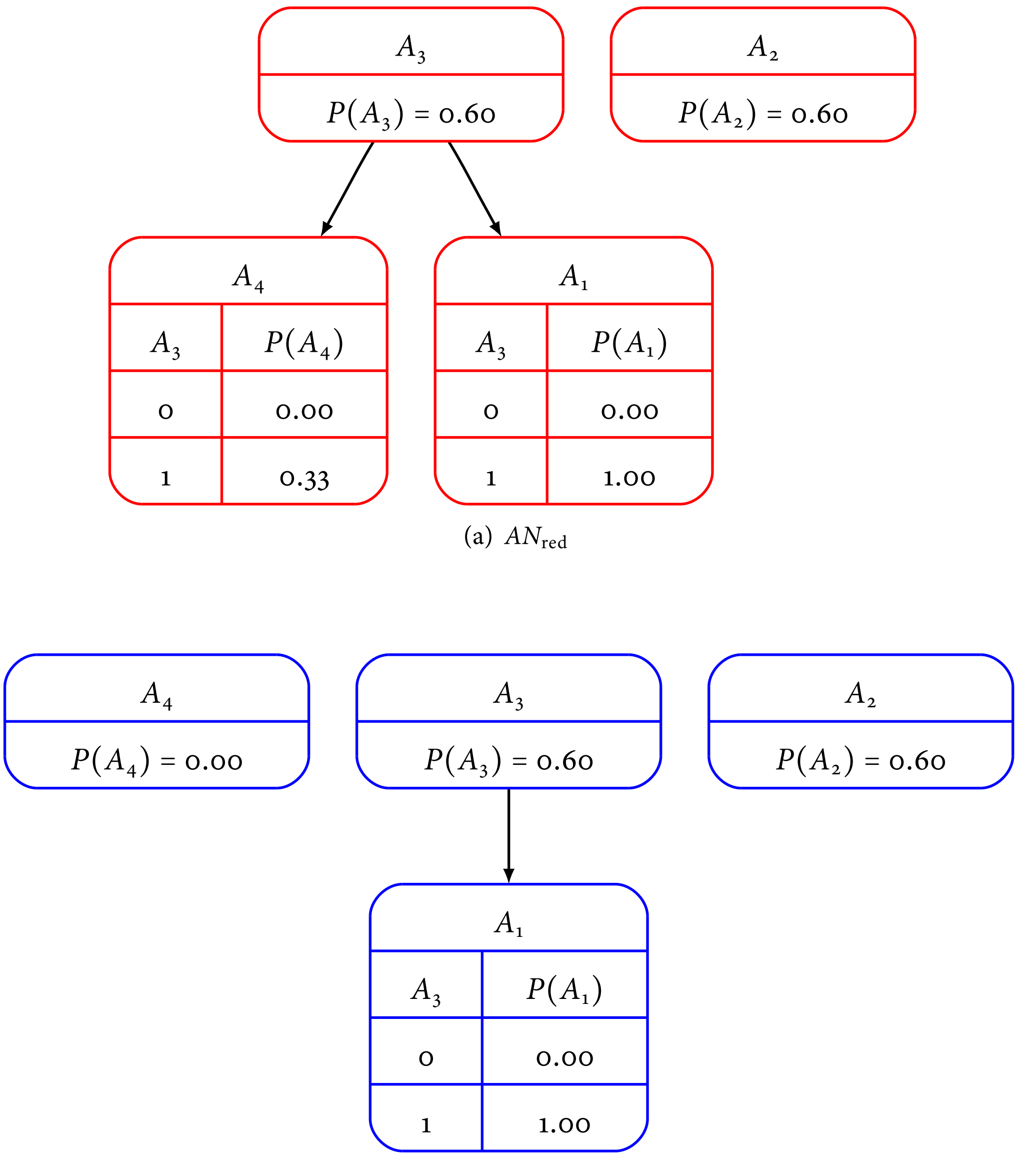
Figure 9.4: Transformed affordance information for the two robots $\mathcal{R}_{\text{red}}$ and $\mathcal{R}_{\text{blue}}$ as detected by $\mathcal{R}_{\text{red}}$. The lines correspond to objects with the columns containing the affordances.

In order to exploit these advantages, $\mathcal{T}^m$ has to be transformed so it can be used as sample data of the random variables in A. This is achieved by restructuring robot $\mathcal{R}_l$'s validation results $b \in \{T, F, \perp\}$ of all the triples in $\mathcal{T}^m_l$ with respect to the objects:

$$\mathcal{T}^{m\prime}_l = \left\{(o, b^l_1, \ldots, b^l_p) \mid b^l_j = Valid_j(I_m(t), o, \mathcal{R}_l)\ \forall (\Lambda_j, o, b^l_j) \in \mathcal{T}^m_l\right\}. \tag{9.5}$$

$\mathcal{T}^{m\prime}_l$ consists of one $(p+1)$-tuple for each object, where $p = |\Lambda|$ is the number of detectable affordances. Each tuple corresponds to one object in the environment of the robots. The transformation of the data in Tab. 9.1 is shown in Fig. 9.4.

In the following, a Bayesian network is called affordance network $AN = (G, \Theta)$ if its nodes are random variables that represent the detected affordances in Λ. *Structural Expectation Maximization* is used to learn the structure and the parameters of the network that best explain the data in $\mathcal{T}^{m\prime}$. It utilizes the *Maximum Likelihood* method to estimate missing data. Fig. 9.5 shows the affordance networks AN_{red} and AN_{blue} that correspond to the data in Fig. 9.4. As can be seen, the *Alternating Model Selection EM algorithm* not only estimates the parameters for the missing data, but also changes the probabilities for the provided data ($P(A_1) = 0.6$ instead of 2/3). This improves the chances of finding the most likely structure and parameters for the provided data [76].



(a) AN_{red}

(b) AN_{blue}

Figure 9.5: Affordance networks for the data in Tab. 9.1

9.6 Comparing affordance networks

The previously described method is then used by each robot to calculate from its filtered perception sequence $I_m(t)$ of robot $\mathcal{R}_m$ the according affordance network AN_i for robot $\mathcal{R}_i$. This is then interpreted as the current behavioral capabilities of $\mathcal{R}_i$. This section describes how two affordance networks can be compared with each other. As stated in Sec. 9.2.1, a Bayesian network $AN = (G, \Theta)$ consists of two components. The *structure component* $G = (A, E)$ is a directed acyclic graph (DAG) with its nodes representing the random variables and its edges the dependencies between them. The *parameter component* Θ is a collection of local interaction models. It describes the probabilities of each variable A_i conditioned on its parent node set $Pa(A_i)$ in G. Both contain causality information between the nodes in AN.

In the following, the metric that calculates the behavioral distance between two robots $\mathcal{R}_1$ and $\mathcal{R}_2$ will be named $D_{AN}(AN_1, AN_2)$. The metric needed to compare two affordance networks has to take into account the structural as well as the parameter component. The individual component metrics will be called $D_{\text{struct}}(AN_1, AN_2)$ and $D_{\text{param}}(AN_1, AN_2)$, respectively.

The following two subsections present the calculation of the structural and parameter difference. Subsequently, the affordance network metric is exemplary applied to the example affordance networks in Fig. 9.5.

9.6.1 Structural difference of affordance networks

Comparing two arbitrary graphs in the general case is NP-complete. For the special case of an affordance network, being a DAG with unique node labeling, Dickinson et al. showed that their Graph Edit Distance algorithm (GED) is able to perform it in polynomial time [67]. It calculates the difference between two graphs as the minimal cost of transforming one graph into the other.

The transformation requires an unique identification of the same nodes and edges in the two graphs that are compared. This is defined by a label representation.

Definition 9.2 (Label representation for graphs with unique node labeling [67]) *The graph*

$$G = (V, E, \alpha, \beta)$$

is called a labeled graph, if the function $\alpha : V \to L_V$ assigns labels to nodes, and the function $\beta : E \to L_E$ assigns labels to edges. The label representation $\mathcal{L}(G)$, is defined by $\mathcal{L}(G) = (L, C, \lambda)$, where

(i) $L = \{\alpha(v) \mid v \in V\}$, *with* $v_1 \neq v_2 \Rightarrow \alpha(v_1) \neq \alpha(v_2)\ \forall\ v_1, v_2 \in V$,

(ii) $C = \{(\alpha(v_1), \alpha(v_2)) \mid (v_1, v_2) \in E\}$, *and*

(iii) $\lambda : C \to L_E$ *with* $\lambda(\alpha(v_1), \alpha(v_2)) = \beta(v_1, v_2)\ \forall (v_1, v_2) \in E$.

In order to transform one graph into the other, the edit operations *changing*, *inserting*, and *removing* a node or edge are needed. Let $G_1 \overset{es}{\to} G_2$ denote the transformation of G_1 into G_2 by a sequence of edit operations $es = op_1, \dots, op_n$. The cost of es is calculated as the sum of the individual costs $c(es) = \sum_{i=1}^{n} c(op_i)$, with $c(\cdot) > 0$. The GED between the two graphs G_1 and G_2 is then calculated as defined in Def. (9.3).

Definition 9.3 (Graph Edit Distance [67]) *Let G_1, G_2 be labeled graphs and $\mathcal{L}(G_1)$, $\mathcal{L}(G_2)$ be the corresponding label representations. The GED between G_1 and G_2 is then calculated as*

$$D_{GED}(G_1, G_2) = |L_1| + |L_2| - 2|L_1 \cap L_2| + |C_1| + |C_2| - 2|C_0| + |C'_0|\,, \tag{9.6}$$

where

$$C_0 = \{(i,j) \mid (i,j) \in C_1 \cap C_2 \wedge \lambda_1(i,j)) = \lambda_2(i,j)\}$$

is the set of ***equally*** *labeled edges and*

$$C'_0 = \{(i,j) \mid (i,j) \in C_1 \cap C_2 \wedge \lambda_1(i,j)) \neq \lambda_2(i,j)\}$$

is the set of ***differently*** *labeled edges between equally labeled nodes of the two graphs.*

Some simplifications are possible when applying the GED metric to affordance networks. Given two affordance networks $AN_1 = (G_1, \Theta_1)$ with $G_1 = (A^1, E^1)$ and $AN_2 = (G_2, \Theta_2)$ with $G_2 = (A^2, E^2)$. Since the GED only compares the graphs' structures, it follows that $D_{GED}(AN_1, AN_2) = D_{GED}(G_1, G_2)$. In addition, some terms in the GED metric are irrelevant in the context of affordance networks, as will be pointed out in the following.

For calculating $D_{\text{struct}}(G_1, G_2)$ based on $D_{GED}(G_1, G_2)$ the affordance networks at first need a label representation, which in turn relies on the labeling functions α and β:

$$\alpha \;:\; A_i \mapsto \Lambda_i, \quad A_i \in A,\ \Lambda_i \in \Lambda \tag{9.7}$$
$$\beta \;:\; (A_i, A_j) \mapsto 1, \quad \forall (A_i, A_j) \in E, i \neq j \tag{9.8}$$

The nodes are naturally labeled uniquely by their corresponding affordances $\Lambda_i \in \Lambda$. The definition of β, assigning to all edges the same value, goes along with the edge semantic and the affordance networks. An edge between two nodes simply states a causal dependency between them. They convey no additional meaning like strength or type of the relationship. To calculate the difference it is only important to know whether an edge does exist or not in the graph.

Definition 9.4 (Label representation for affordance networks) *The label representation*

$$\mathcal{L}(G) = (L, C, \lambda)$$

of the graph component of an affordance network $AN = (G, \mathbf{\Theta})$ is defined by:

(i) $L = \{\Lambda_1, \dots, \Lambda_n\}$, $n = |\Lambda|$, $A_1 \neq A_2 \Rightarrow \alpha(A_1) \neq \alpha(A_2)\ \forall\ A_1, A_2 \in A$

(ii) $C = \{(\alpha(A_i), \alpha(A_j)) \mid (A_i, A_j) \in E\}$

(iii) $\lambda : C \to 1$ *with* $\lambda(\alpha(A_i), \alpha(A_j)) = \beta(A_i, A_j) = 1\ \forall\ (A_i, A_j) \in E,\ i \neq j.$

Two nodes $A_i \in G_1$ and $A_j \in G_2$ are called corresponding if they are assigned the same label: $\alpha_1(A_i) = \alpha_2(A_j)$.

When a robot is monitoring other robots in order to recognize affordances, it checks its observations for the same set of affordances for each robot. Therefore, it can ensure that all the affordance networks, which it has created for the different robots, contain the same set of nodes with the same labels, hence $|L_1| + |L_2| - 2|L_1 \cap L_2| = 0$. The second simplification follows from the definition of β. When two graphs contain the same edge, this edge is also labeled the same. Consequently, the term $|C'_0|$, which counts all edges with different labels, can be omitted from the metric. This leads to the following distance measurement for affordance networks:

$$D_{\text{struct}}(G_1, G_2) = D_{\text{struct}}(AN_1, AN_2) = |C_1| + |C_2| - 2|C_0| \tag{9.9}$$

The following example calculates the structural distance between the affordance networks AN_{red} and AN_{blue} from Fig. 9.5. The label representations are defined by $\mathcal{L}(G_{\text{red}})$ and $\mathcal{L}(G_{\text{blue}})$:

- $\mathcal{L}(G_{\text{red}})$
  - $L_{\text{red}} = \{\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4\}$
  - $C_{\text{red}} = \{(\Lambda_3, \Lambda_4), (\Lambda_3, \Lambda_1)\}$
  - $\lambda_{\text{red}} : (x, y) \to 1 \ \forall (x, y) \in C_{\text{red}}$
- $\mathcal{L}(G_{\text{blue}})$
  - $L_{\text{blue}} = \{\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4\}$
  - $C_{\text{blue}} = \{(\Lambda_3, \Lambda_1)\}$
  - $\lambda_{\text{blue}} : (x, y) \to 1 \ \forall (x, y) \in C_{\text{blue}}$

Both graphs have one similar edge, so that $C_0 = \{(\Lambda_3, \Lambda_1)\}$. The structural distance AN_{red} and AN_{blue} is:

$$D_{\text{struct}}(AN_{\text{red}}, AN_{\text{blue}}) = |C_{\text{red}}| + |C_{\text{blue}}| - 2|C_0| = 2 + 1 - 2 = 1$$

9.6.2 Parameter difference of affordance networks

The basic idea behind the calculation of the difference between two affordance networks is to interpret the networks' nodes as points in the same metric space and to use the Manhattan metric. The distance sum of all the corresponding nodes from AN_1 and AN_2 is then the parameter difference between the two networks.

To support the description of the parameter difference, some definitions have to be introduced first.

Definition 9.5 (Event combination of the parent nodes) *Let $AN = (G, \Theta)$ and A_i be a node from $G = (A, E, \alpha, \beta)$. Let $Pa(A_i)$ be the set of A_i's parent nodes and*

$$F(A_i) = F_i = \left\{ f^i_{jp} = (\Lambda_j, p) \mid \Lambda_j = \alpha(A_j),\ A_j \in Pa(A_i),\ p \in \{0, 1\} \right\} \tag{9.10}$$

the event set of the A_i's parent affordance random variables, $Pa(A_i)$, with $p = 1$ indicating that affordance Λ_j is provided and $p = 0$ otherwise. The event combination set of these variables is then defined as

$$\begin{aligned}\boldsymbol{ec}(A_i) = \boldsymbol{ec}_i = \Big\{ec_{i,k} = (f^i_{j_1 p_1}, \ldots, f^i_{j_l p_l}) \ \mid f^i_{j_m q_m} &\in F(A_i),\\ j_u &\neq j_v \ \forall\, u \neq v,\\ ec_{iw} &\neq ec_{ix} \ \forall\, w \neq x,\\ l = |Pa(A_i)|, \ k &\in \{1, \ldots, 2^l\}\Big\} \ . \end{aligned} \tag{9.11}$$

As the possible events for a random variable are either true or false, $|F_i| = 2|Pa(A_i)|$ and $|\boldsymbol{ec}_i| = 2^{|Pa(A_i)|}$.

Consider node A_1 of the affordance network AN_2 in Fig. 9.6(b) as an example. It has two parents A_2 and A_3. In this case,

$$F(A_1) = \{(\Lambda_2, 1), (\Lambda_2, 0), (\Lambda_3, 1), (\Lambda_3, 0)\} \tag{9.12}$$

and

$$\begin{aligned}\boldsymbol{ec}(A_1) = \{ \ &((\Lambda_2, 1), (\Lambda_3, 1)) \ ,\\ &((\Lambda_2, 1), (\Lambda_3, 0)) \ ,\\ &((\Lambda_2, 0), (\Lambda_3, 1)) \ ,\\ &((\Lambda_2, 0), (\Lambda_3, 0))\} \ . \end{aligned} \tag{9.13}$$

With the help of the elements of $\boldsymbol{ec}_i$, it is possible to construct a coordinate system. Within this coordinate system, it is possible to compare the parameter difference of an affordance for different robots.

Definition 9.6 (Point representation of a node) *Let A_i be a node in $G = (A, E, \alpha, \beta)$ and $\boldsymbol{ec}_i$ defined according to Def. (9.5). The set*

$$coord(A_i) = \{coord(A_i)_k = (ec_{i,k}, \Theta_{i,k}) \mid \ P(A_i \,|\, ec_{i,k}) = \Theta_{i,k}\} \tag{9.14}$$

is the coordinate set of $point(A_i)$ corresponding to node A_i.

Let further the k-th axis of $coord(A_i)$ be

$$axis(A_i)_k = ec_{i,k} \tag{9.15}$$

and denote its value as

$$value(A_i)_k = \Theta_{i,k} \ . \tag{9.16}$$

The set $coord(A_i)$ contains a coordinate k for each event combination $ec_{i,k}$ and therefore has $2^{|Pa(A_i)|}$ coordinates.

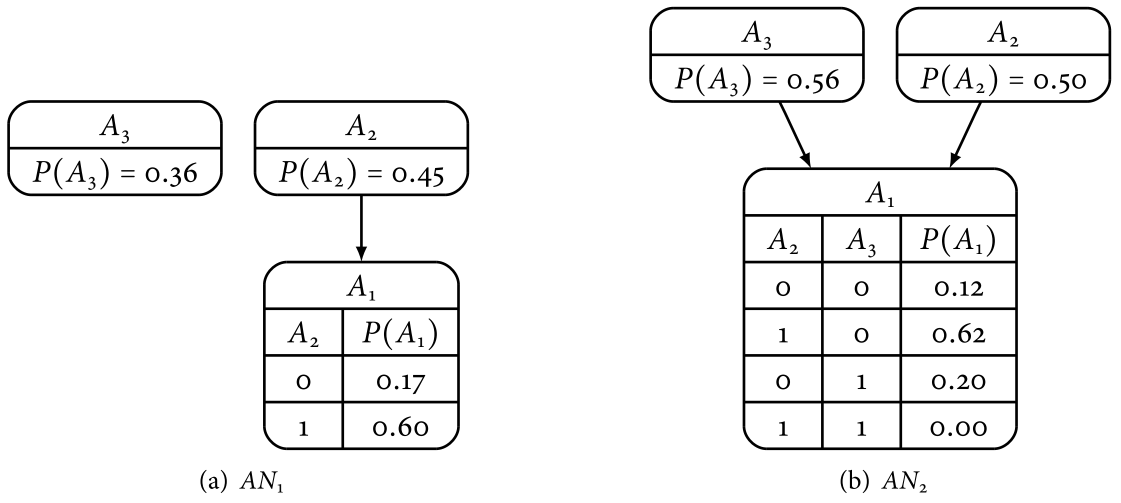


Figure 9.6: Graph AN_1 is lacking edge $A_3 \rightarrow A_1$

Definition 9.7 (Comparability of points) *Let $A_i^{G_1} \in G_1$ and $A_n^{G_2} \in G_2$ be two corresponding nodes for the graphs of different robots, $\alpha(A_i^{G_1}) = \alpha(A_n^{G_2})$, and $coord(A_i^{G_1})$ and $coord(A_n^{G_2})$ their respective coordinate sets. The points reside in the same coordinate system if for all $coord(A_i^{G_1})_k \in coord(A_i^{G_1})$ exactly one coordinate $coord(A_n^{G_2})_{k'} \in coord(A_n^{G_2})$ with $axis(A_i^{G_1})_k = axis(A_n^{G_2})_{k'}$ exists. In this case, the nodes are said to be* comparable.

For two *comparable* nodes, $A_i^{G_1}$ and $A_n^{G_2}$, a parameter distance can be calculated by Eq. (9.17):

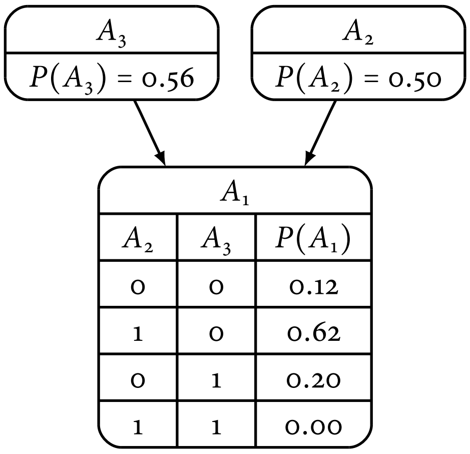
$$D_{\text{param}}(A_i^{G_1}, A_n^{G_2}) = \sum_{\substack{k=1 \\ axis(A_i^{G_1})_k = axis(A_n^{G_2})_{k'}}}^{2^l} \left| value(A_i^{G_1})_k - value(A_n^{G_2})_{k'} \right| \tag{9.17}$$

Affordance networks do not necessarily contain only comparable nodes. Figures 9.6, 9.7, and 9.8 show three affordance networks AN_1, AN_2, and AN_3, which cannot be compared with each other. The rest of this section explains how to cope with this incomparability.

In all of these three cases, the coordinate sets of the points must be extended so that they satisfy Def. (9.7). This is supported by the Markov property, which guarantees that two random variables of an affordance network are independent if they are not directly connected by an edge. The law of conditional independence allows to extend the condition set $ec_{i,k}$ of probability $P(A_i \mid ec_{i,k})$ by additional conditions without changing the probability, if A_i is independent of the additional conditions.

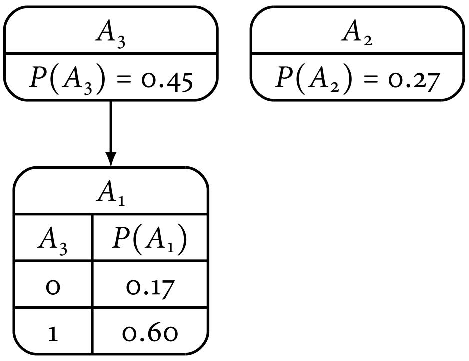
Definition 9.8 (Extended event combination of the parent nodes) *Let $A_i^{G_1}$ and $A_n^{G_2}$ be two corresponding nodes in different affordance networks, $Pa(A_i^{G_1})$ and $Pa(A_n^{G_2})$ their individual parent sets, and $\Lambda^{ext} = \{\alpha(A_k) \,|\, A_k \in Pa(A_i^{G_1}) \cup Pa(A_n^{G_2})\}$ be the set of both parents' affordances. The extended event set of both nodes, $A_i^{G_1}$ and $A_n^{G_2}$, is given by*

$$F^{ext}(A_i^{G_1}, A_n^{G_2}) = F_{in}^{ext} = \left\{ f_{jp}^{in} = (\Lambda_j, p) \mid \Lambda_j \in \Lambda^{ext},\ p \in \{0,1\} \right\} . \tag{9.18}$$



A_2	A_3	$P(A_1)$
0	0	0.12
1	0	0.62
0	1	0.20
1	1	0.00

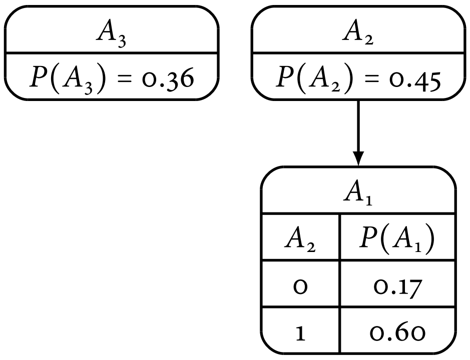
(a) AN_2 from Fig. 9.6(b)



A_3	$P(A_1)$
0	0.17
1	0.60

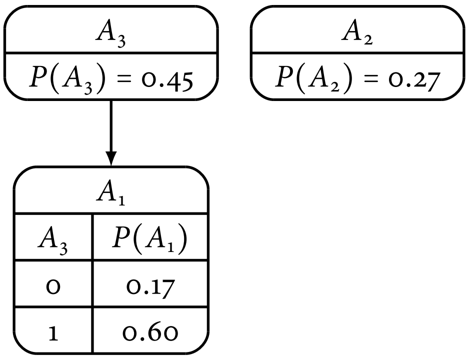
(b) AN_3

Figure 9.7: Opposite situation: graph AN_3 is lacking edge $A_2 \rightarrow A_1$



A_2	$P(A_1)$
0	0.17
1	0.60

(a) AN_1 from Fig. 9.6(a)



A_3	$P(A_1)$
0	0.17
1	0.60

(b) AN_3 from Fig. 9.7(b)

Figure 9.8: Both affordance networks are lacking an edge that exists in the other network.

The set of event combinations of $A_i^{G_1}$'s parent nodes, extended by the missing parents of $A_n^{G_2}$, is then given by

$$\boldsymbol{ec}^{ext}(A_i^{G_1}, A_n^{G_2}) = \boldsymbol{ec}_{in}^{ext} = \Big\{ ec_{in,k}^{ext} = (f_{j_1p_1}^{in}, \ldots, f_{j_lp_l}^{in}) \mid f_{j_mq_m}^{i} \in F_{in}^{ext}, \\ j_u \neq j_v \ \forall\, u \neq v, \\ ec_{in,w}^{ext} \neq ec_{in,x}^{ext} \ \forall\, w \neq x, \\ l = |\Lambda^{ext}|, k \in \{1, \ldots, 2^l\} \Big\} . \tag{9.19}$$

Definition 9.9 (Probability of affordances with extended event combinations) *Let two nodes, $A_i^{G_1} \in G_1$ and $A_n^{G_2} \in G_2$, be corresponding but not comparable as they do not show the properties of Def. (9.7). Let $\boldsymbol{ec}_i$ be the set of event combinations of node $A_i^{G_1}$ and $\boldsymbol{ec}_{in}^{ext}$ the set of event combinations extended by $A_n^{G_2}$'s event combination set, which emerged from $\boldsymbol{ec}_i$.*

With the independence definition and the conditioned independence, the extended event combinations condition the same probability as the original event combination:

$$P(A_i^{G_1} \mid ec_{in,k_p}^{ext}) = P(A_i^{G_1} \mid ec_{i,k}) \ \ \forall\ ec_{in,k_p}^{ext} \in \boldsymbol{ec}_{in}^{ext},\ ec_{i,k} \in \boldsymbol{ec}_i,\ p \in \{0,1\} \tag{9.20}$$

Each component in the original condition set, $f_{j_up_u}^{i} \in ec_{i,u} = (j_{j_1p_1}^{i}, \ldots, f_{j_lp_l}^{i})$, $l = |Pa(A_i)|$, $u = k$, then has a corresponding component in the extended condition set, $f_{j_vp_v}^{in} \in ec_{in,v}^{ext} = (j_{j_1p_1}^{in}, \ldots, f_{j_{l'}p_{l'}}^{in})$, $l' = |\Lambda^{ext}|$, $v = k_p$, with $f_{j_up_u}^{in} = f_{j_vp_v}^{i}$. With Def. (9.9) the following equality holds: $P(A_i^{G_1} \mid ec_{in,v}^{ext}) = P(A_i^{G_1} \mid ec_{i,u})$.

Using the equality of the probability distribution, the extended coordinate representation of the point corresponding to $A_i^{G_1}$ can be defined.

Definition 9.10 (Extended point representation of a node) *Let $A_i^{G_1} \in G_1$ and $A_n^{G_2} \in G_2$ be two corresponding nodes violating the properties of Def. (9.7). Let $\boldsymbol{ec}_{in}^{ext}$ be the extended event combination of the parent nodes of $A_i^{G_1}$. The extended coordinate set $coord_{in}^{ext}(A_i^{G_1})$ of the point corresponding to $A_i^{G_1}$ can then be defined as:*

$$coord_{in}^{ext}(A_i^{G_1}) = \left\{ (ec_{in,k}^{ext}, \Theta_{i,k}) \mid ec_{in,k}^{ext} \in \boldsymbol{ec}_{in}^{ext},\ \Theta_{i,k} = P(A_i^{G_1} \mid ec_{in,k}^{ext}) \right\} \tag{9.21}$$

The k-th axis of $coord_{in}^{ext}(A_i^{G_1})$ is then defined as

$$axis(A_i^{G_1})_k = ec_{in,k}^{ext} \tag{9.22}$$

and its value denoted as

$$value(A_i^{G_1})_k = \Theta_{i,k} . \tag{9.23}$$

$coord_{in}^{ext}(A_i^{G_1})$ is also called the extended point representation of node $A_i^{G_1}$.

Consider the two graphs AN_1 and AN_2 in Fig. 9.6. In order to calculate the distance between them, $A_1^{G_1}$, the node A_1 in graph AN_1, has to be extended so that it can be compared to $A_1^{G_2}$, the

A_1	
A_2	$P(A_1)$
0	0.17
1	0.60

(a) A_1 from Fig. 9.6(a)

A_1		
A_2	A_3	$P(A_1)$
0	0	0.17
0	1	0.17
1	0	0.60
1	1	0.60

(b) A_1 after the extension

Figure 9.9: Extended point representation of node A_1 from Fig. 9.6(a): after extending the node's condition set, it can be compared to A_1 from graph AN_2 in Fig. 9.6(b)

node A_1 in graph AN_2. This is done by firstly determining $F^{ext}(A_1^{G_1}, A_1^{G_2})$:

$$\begin{aligned} F^{ext}(A_1^{G_1}, A_1^{G_2}) = \{ & f_{21}^{11} = (\Lambda_2, 1), \\ & f_{20}^{11} = (\Lambda_2, 0), \\ & f_{31}^{11} = (\Lambda_3, 1), \\ & f_{30}^{11} = (\Lambda_3, 0)\} \end{aligned} \tag{9.24}$$

Using this extended event set, the extended event combination of the parent nodes is given by

$$\begin{aligned} \boldsymbol{ec}^{ext}(A_1^{G_1}, A_1^{G_2}) = \{ & (f_{21}^{11}, f_{31}^{11}), \\ & (f_{21}^{11}, f_{30}^{11}), \\ & (f_{20}^{11}, f_{31}^{11}), \\ & (f_{20}^{11}, f_{30}^{11})\} \\ = \{ & ((\Lambda_2, 1), (\Lambda_3, 1)), \\ & ((\Lambda_2, 1), (\Lambda_3, 0)), \\ & ((\Lambda_2, 0), (\Lambda_3, 1)), \\ & ((\Lambda_2, 0), (\Lambda_3, 0))\} \ . \end{aligned} \tag{9.25}$$

The extended point representations are made up by the following equations and visualized in Fig. 9.9.

$$\begin{aligned} P(A_1^{G_1} \mid ec_{11,1_0}^{ext}) &= P(A_1^{G_1} \mid ec_{11}) = 0.17 \\ P(A_1^{G_1} \mid ec_{11,1_1}^{ext}) &= P(A_1^{G_1} \mid ec_{11}) = 0.17 \\ P(A_1^{G_1} \mid ec_{11,2_0}^{ext}) &= P(A_1^{G_1} \mid ec_{12}) = 0.6 \\ P(A_1^{G_1} \mid ec_{11,2_1}^{ext}) &= P(A_1^{G_1} \mid ec_{12}) = 0.6 \end{aligned}$$

Let $c_i = coord_{in}^{ext}(A_i^{G_1})$ and $c_n = coord_{ni}^{ext}(A_n^{G_2})$ be the extended point representations of the corresponding nodes $A_i^{G_1}$ and $A_n^{G_2}$. The Manhattan distance is calculated as follows:

$$D_{\text{param}}(A_i^{G_1}, A_n^{G_2}) = \sum_{\substack{k=1 \\ axis(A_i^{G_1})_k = axis(A_n^{G_2})_{k'}}}^{2^{|\Lambda^{ext}|}} |value(A_i^{G_1})_k - value(A_n^{G_2})_{k'}| \qquad (9.26)$$

This leads to the calculation of the Manhattan distance of two affordance networks:

$$D_{\text{param}}(AN_1, AN_2) = D_{\text{param}}(G_1, G_2) = \sum_{\substack{A_i^{G_1} \in G_1, A_n^{G_2} \in G_2 \\ \alpha_1(A_i^{G_1}) = \alpha_2(A_n^{G_2})}} D_{\text{param}}(A_i^{G_1}, A_n^{G_2}) \qquad (9.27)$$

D_{param} applied to the networks from Fig. 9.5 (page 111) results in the following parameter distance:

$$\begin{aligned} D_{\text{param}}(AN_{\text{red}}, AN_{\text{blue}}) &= \sum_{i=1}^{4} D_{\text{param}}(A_i^{\text{red}}, A_i^{\text{blue}}) \\ &= 0.0 + 0.0 + 0.0 + 0.33 \\ &= 0.33 \end{aligned} \qquad (9.28)$$

9.6.3 Affordance network distance metric

With the definition of both distance components, the total distance of the affordance networks can be calculated as the weighted sum of both components:

$$D_{AN}(AN_1, AN_2) = \eta \cdot D_{\text{struct}}(AN_1, AN_2) + (1 - \eta) \cdot D_{\text{param}}(AN_1, AN_2)\ . \qquad (9.29)$$

The total distance of the example networks AN_{red} and AN_{blue} in Fig. 9.5 with $\eta = 0.5$ then is

$$\begin{aligned} D_{AN}(AN_{\text{red}}, AN_{\text{blue}}) &= 0.5 \cdot D_{\text{struct}}(AN_{\text{red}}, AN_{\text{blue}}) + 0.5 \cdot D_{\text{param}}(AN_{\text{red}}, AN_{\text{blue}}) \\ & 0.5 \cdot 1 + 0.5 \cdot 0.33 = 0.665\ . \end{aligned} \qquad (9.30)$$

If robot $\mathcal{R}_m$ in a robot group $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$ has observed other robots, detected affordances, and constructed the corresponding affordance networks for all the other robots and for itself, it can then choose to imitate robot $\mathcal{R}_{imitate}$ that is most similar to itself in terms of the affordance network metric:

$$\mathcal{R}_{imitate} = \underset{\mathcal{R}_i \in \boldsymbol{\mathcal{R}},\ \mathcal{R}_i \neq \mathcal{R}_m}{\arg\min} \{D_{AN}(AN_i, AN_m)\} \qquad (9.31)$$

9.7 Evaluation

The following three experiments evaluate the applicability and robustness of the demonstrator selection. The starting situation in all of these experiments is an imitator robot that has to choose one imitatee from a group of potential demonstrator robots. The first experiment shows detailed how different capabilities lead to different affordance observations of which in turn different

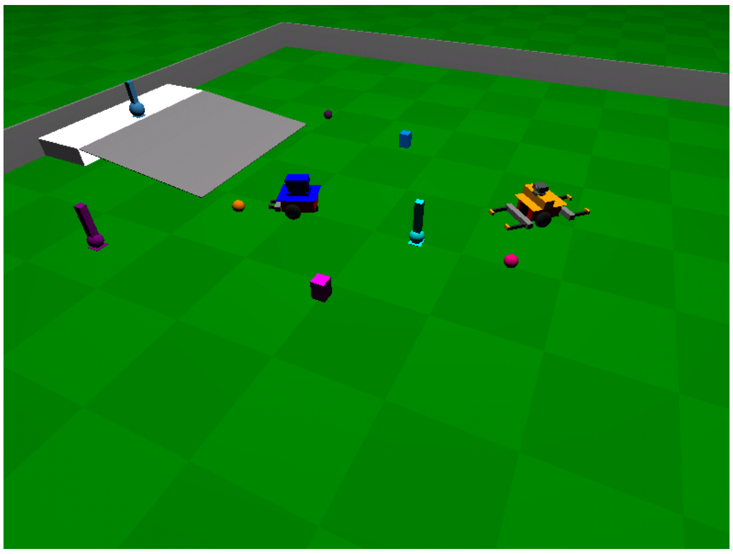


Figure 9.10: Evaluation environment containing morphologically different robots and objects of different shape, width, height, mass, and surface

affordance networks are generated. In the second experiment, the imitation performance and robustness of the whole approach is analyzed. Both experiments assume the variance of the objects' properties to support the generation of meaningful statistics of the recognized affordances. With increasing variance, the usefulness of the created affordance networks decreases as the same affordance is averaged over objects of greatly varying properties. This is the case if, e. g., the imitator has observed a demonstrator being able to push a lightweight object but unable to push a heavy one. The last experiment shows how to overcome this problem by clustering objects according to their properties.

9.7.1 Experimental setup

Multiple morphologically different robots are located in a Gazebo environment containing objects of different sizes and shapes. Fig. 9.10 shows an example environment with two of those robots with different morphology and objects of different sizes and shapes. In the exploration phase, the robots are interacting with those objects and thereby detecting the affordances offered to them. In addition, they are continuously monitoring the other robots in the environment and thereby detecting affordances of those robots as well. Whenever a robot decides to imitate another robot, it executes the demonstrator selection algorithm and imitates the selected robot. The remainder of this section presents the used parameter intervals of the robots' and objects' properties

9.7.1.1 Parameterization of the environment

The Pioneer2DX is used as the base robot platform, which already exists as a module for the Gazebo simulation environment [25]. Two simulated SickLMS 200 laser scanners [23] are cou-

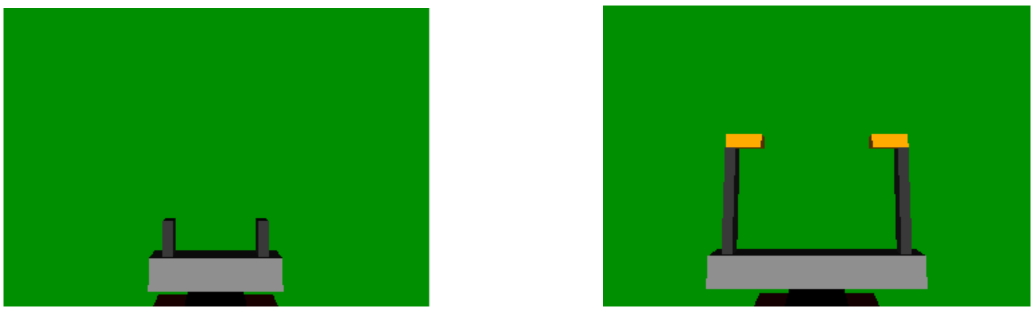

(a) Standard gripper (b) Barb gripper

Figure 9.11: Two different grippers that lead to very different affordances of the corresponding robots

Table 9.2: Parametrization of the robots used in the experiments

parameter		values	description
motor	power	$[0.3, 6.0]\ kg$	maximal weight a robot can pull/push
	speed	$[0.03, 0.2]\ m/s$	controls the impulse a robot impact on an object
gripper	length	$[0.08, 0.2] m$	the longer the gripper the deeper the objects can be
	span	$[0.16, 0.5]\ m$	limits the diameter of objects that can be gripped
	closing force	$[1.0, 30.0]\ kg$	controls the contact pressure (to pull heavier objects the closing force must be higher)
	lifting force	$[30.0, 80.0]\ kg$	controls the friction (to lift heavier objects the closing force must be higher)
	form	$\{normal, barb\}$	different forms lead to different interaction possibilities (Fig. 9.11)

pled to scan the full 360° of the robot's environment for objects and other robots. The lasers return the ID and the relative position for both. The robot knows the objects' individual properties. In all experiments, the robots have the same perception configuration. The evaluation concentrates on the morphological differences that are leading to different action capabilities, which are the movement and gripping capabilities. Tab. 9.2 lists the different parameters and shows the intervals of their values together with their effect on the robot's action capabilities.

Not only the robot's morphology determines, which affordances are offered by an object, but also the properties of that object itself. The evaluation experiments used objects of different shape, width, height, mass, and surface. Tab. 9.3 provides a detailed overview.

Table 9.3: Parametrization of the objects used in the experiments

parameter	values	discretization
mass	$[1.0, 5.0]\ kg$	$0.5\ kg$
width	$[0.04, 0.24]\ m$	$0.05\ m$
height	$[0.17, 0.2]\ m$	$0.05\ m$
friction	$[50, 100]$ %	0.1 %
shaped	{ sphere, cube, cylinder}	

9.7.1.2 Affordances and their validation

As already pointed out above, the algorithm relies on a fixed set of predefined affordances. In all the experiments, the used affordance set was $\Lambda = \{\Lambda_1 \equiv$ seizable, $\Lambda_2 \equiv$ liftable, $\Lambda_3 \equiv$ pushable, $\Lambda_4 \equiv$ pullable $\}$. The affordances are defined as follows:

seizable The robot is able to reach the object, position it in its gripper, and close the gripper with the object in it.

liftable The object can be gripped by the robot. When lifting the gripper with the object in it, it stays fixed to the gripper and is also lifted.

pushable When the robot bumps against the object and continues to move forward, the object pushed forward.

pullable While the robot is driving backwards having previously gripped an object, the object stays within the gripper.

The examination of an affordance $\Lambda_j \in \Lambda$ is conducted by means of $Valid_j(I_m(t), o, \mathcal{R})$ as defined in Sec. 9.2. This function needs a list of preconditions that have to be met in order to test the final condition. The final condition determines the affordance (cf. Sec. 9.4). For the previously presented four affordances, those conditions are represented as finite state machines in Fig. 9.12.

The presented parameters of the robots and objects allow for sufficiently different scenarios to evaluate the demonstrator selection approach in this chapter. Before moving on to the experiments the measurement of imitation success has to be defined.

9.7.1.3 Imitated behavior and how to measure its success

The robots imitate other robots that are performing different skills on the diverse objects in the environment. While doing so, the imitator's error functions of the involved skills will provide

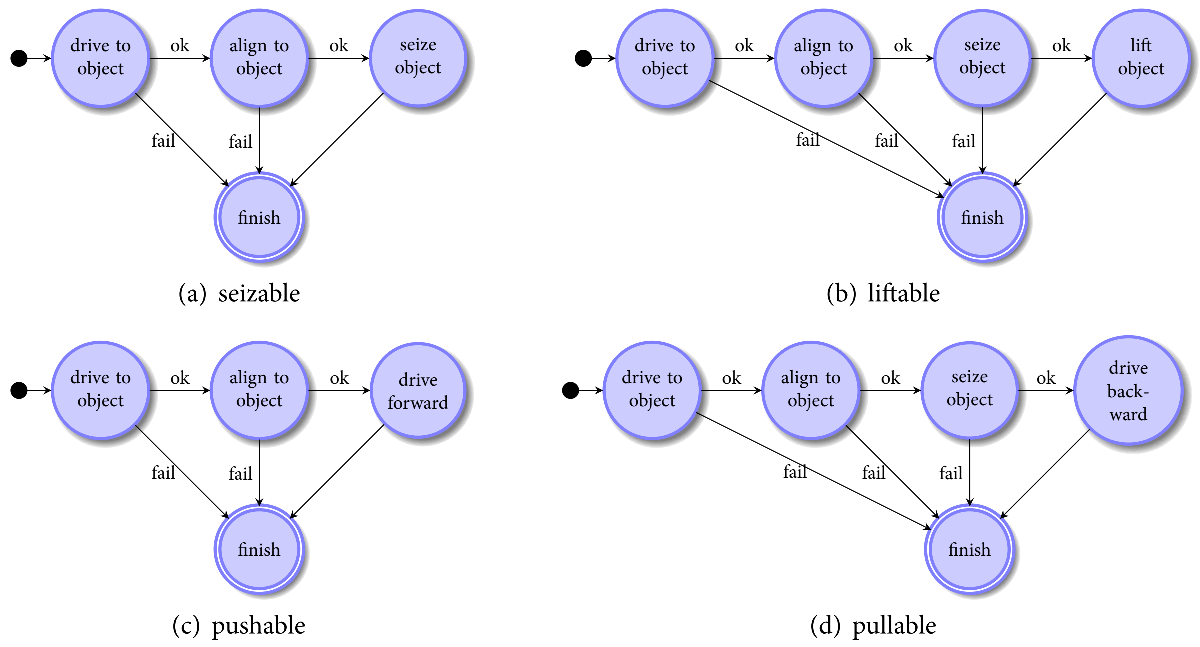


Figure 9.12: Affordance testing conditions modeled as finite state machines for the affordances used in the experiments

feedback regarding how successful the individual skills have been executed. The number of failure signals the strategy layer retrieved while executing the imitated behavior serves as an indicator how wise the demonstrator choice had been.

9.7.2 Selection experiment

The purpose of this experiment is to show the overall feasibility of the demonstrator selection algorithm by means of a complex scenario.

9.7.2.1 Scenario

The environment contains three robots $\boldsymbol{\mathcal{R}} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$ and nine objects $\mathbf{O} = \{o_1, \ldots, o_9\}$. Of the three robots, $\mathcal{R}_1$ is the imitator that has to decide which of the other robots is most similar to itself. All robots are able to explore those objects to find out which of the affordances $\Lambda = \{$ seizable, liftable, pushable, pullable $\}$ are offered by them. The exploration is conducted by means of a set of predefined behaviors.

The concrete parameterization of the robots and objects is shown in Tab. 9.4 and 9.5. As a close look to Tab. 9.4 reveals, robot $\mathcal{R}_1$ seems to resemble more $\mathcal{R}_2$ than $\mathcal{R}_3$. The affordance network metric should therefore return a smaller distance to $\mathcal{R}_2$ than to $\mathcal{R}_3$ for robot $\mathcal{R}_1$. Compared to the object parameter intervals in Tab. 9.3 the objects in this experiment are very similar. They are all designed to be small and lightweight.

Table 9.4: Dimensioning of the three robots in the selection experiment

robots	motor	gripper		
	power	length	power	form
$\mathcal{R}_1$	strong	long	weak	barb
$\mathcal{R}_2$	medium	medium	weak	barb
$\mathcal{R}_3$	weak	short	strong	normal

Table 9.5: Dimensioning of objects in the selection experiment

object	mass	width	height	friction	form
o_1	1.5kg	0.04m	0.41m	95%	cube
o_2	1.5kg	0.04m	0.42m	100%	cube
o_3	1.0kg	0.06m	0.41m	95%	cube
o_4	1.8kg	0.04m	0.42m	95%	cube
o_5	1.5kg	0.06m	0.40m	100%	cylinder
o_6	1.5kg	0.04m	0.41m	100%	cylinder
o_7	1.0kg	0.06m	0.42m	95%	cube
o_8	1.3kg	0.04m	0.39m	100%	cylinder
o_9	1.3kg	0.31m	0.31m	90%	sphere

9.7.2.2 Procedure

The robots start exploring the objects without any information regarding the interaction possibilities. The exploration phase stops after each robot has investigated each object and created its affordance networks. Finally, the affordance network distances $D_{AN}(\mathcal{R}_1, \mathcal{R}_2)$ and $D_{AN}(\mathcal{R}_1, \mathcal{R}_3)$ are calculated.

9.7.2.3 Result

Fig. 9.13, 9.14, and 9.15 show the affordance networks, which robot $\mathcal{R}_1$ has built for all robots including itself based on its observation. The outcome of the affordance network metric $D_{AN} = \eta \cdot D_{\text{struct}}(AN_1, AN_2) + (1-\eta) \cdot D_{\text{param}}(AN_1, AN_2)$ in Eq. (9.29) applied to these networks is shown in Tab. 9.6 for different values of η.

Table 9.6: Behavioral distance calculated by the affordance network metric

η	$D_{AN}(\mathcal{R}_1, \mathcal{R}_2)$	$D_{AN}(\mathcal{R}_1, \mathcal{R}_3)$
0.1	0.41	3.45
0.25	0.34	3.53
0.5	0.23	3.69
0.75	0.11	3.85
0.9	0.045	3.94

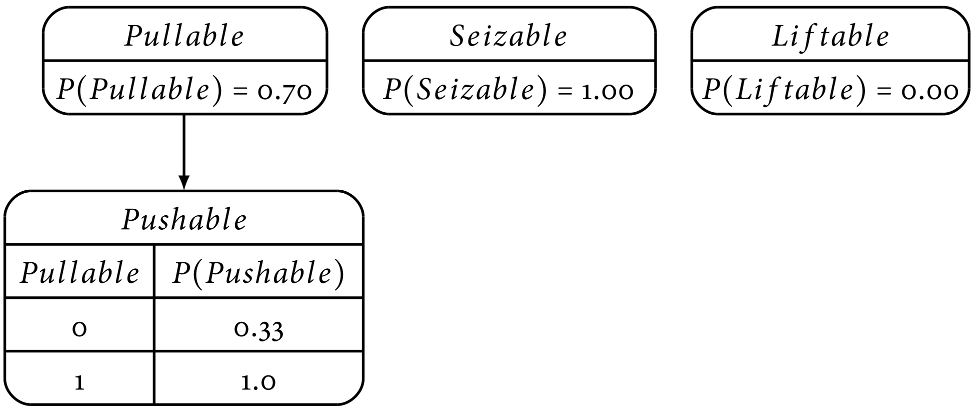


Figure 9.13: Affordance network of robot $\mathcal{R}_1$ (imitator)

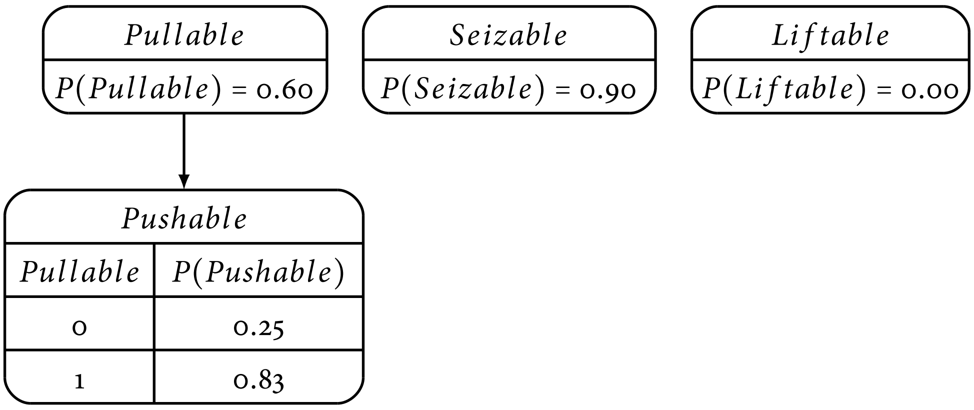


Figure 9.14: Affordance network of robot $\mathcal{R}_2$ (demonstrator)

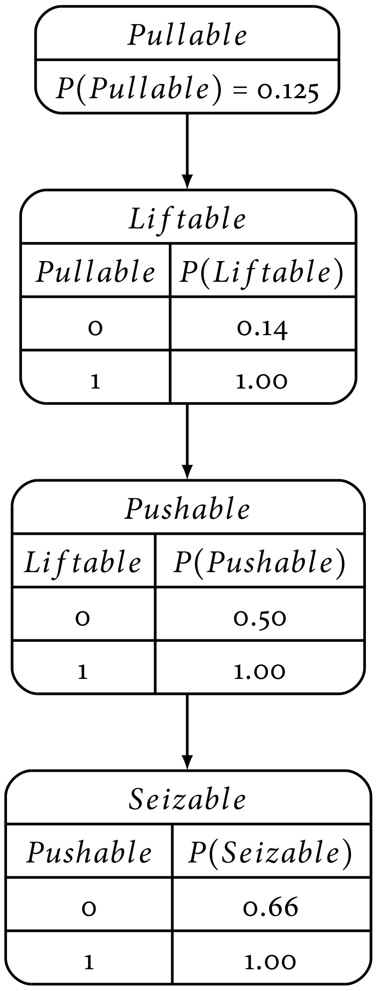


Figure 9.15: Affordance network of robot $\mathcal{R}_3$ (demonstrator)

It shows that the imitator robot $\mathcal{R}_1$ would always choose $\mathcal{R}_2$ for imitation, which is the more similar demonstrator. This indicates that the more similar robots have been offered the more similar affordances by the objects. The affordance networks consequently reflected this similarity, which led to a smaller value in the distance measurement.

9.7.3 Robustness experiment

Imitation in general is of most utility to the robot when it has not already explored much compared to the other robots. Ironically, at this stage, it also has not collected that much information regarding the affordances offered to it and to the other robots. This experiment evaluates how the demonstrator selection algorithm copes with the situation. In addition, it investigates how the algorithm reacts to uncertainty in the collected affordance data. For that purpose, the collected affordance data will be set to "$\perp$" (missing) by 0%, 10%, 20%, and 35%, respectively. The *Alternating Model Selection EM algorithm* by Friedman [76] basically allows for uncertainty in the data. This experiment evaluates, how well it does so in the affordance network context.

9.7.3.1 Scenario

The scenario consists of the three robots as in the previous experiment (Sec. 9.7.2) plus another demonstrator robot $\mathcal{R}_4$. That one is dissimilar to the imitator, representing a bad decision. The objects are also the same as in the previous experiment. Throughout this experiment the metric has equal weights for both components ($\eta = 0.5$).

9.7.3.2 Procedure

One experiment run consists of 35 episodes. Starting with the first episode, the imitator $\mathcal{R}_1$ has no affordance information ($\mathcal{T}_1 = \varnothing$), whereas all demonstrators ($\mathcal{R}_{2-4}$) have already explored the environment. The imitator gets the chance to test one not yet tested random affordance on an object. This populates $\mathcal{T}_1$ by one affordance tuple. The imitator then calculates AN_1 and has to select a demonstrator based on the calculated distances. AN_{2-4} are fix throughout the whole experiment.

Initially, this will obviously be rather random with only little affordance information. Based on the choice, the imitator imitates the demonstrator and immediately applies the imitated behavior sequence.

9.7.3.3 Result

The results are shown in Fig. 9.16. They are displayed relatively to the number of failure signals received in the first episode. As expected, the failure signal for the random imitation experiment stays the same throughout the experiment run. This is the reference performance, marking the case of imitation without using the demonstrator selection algorithm.

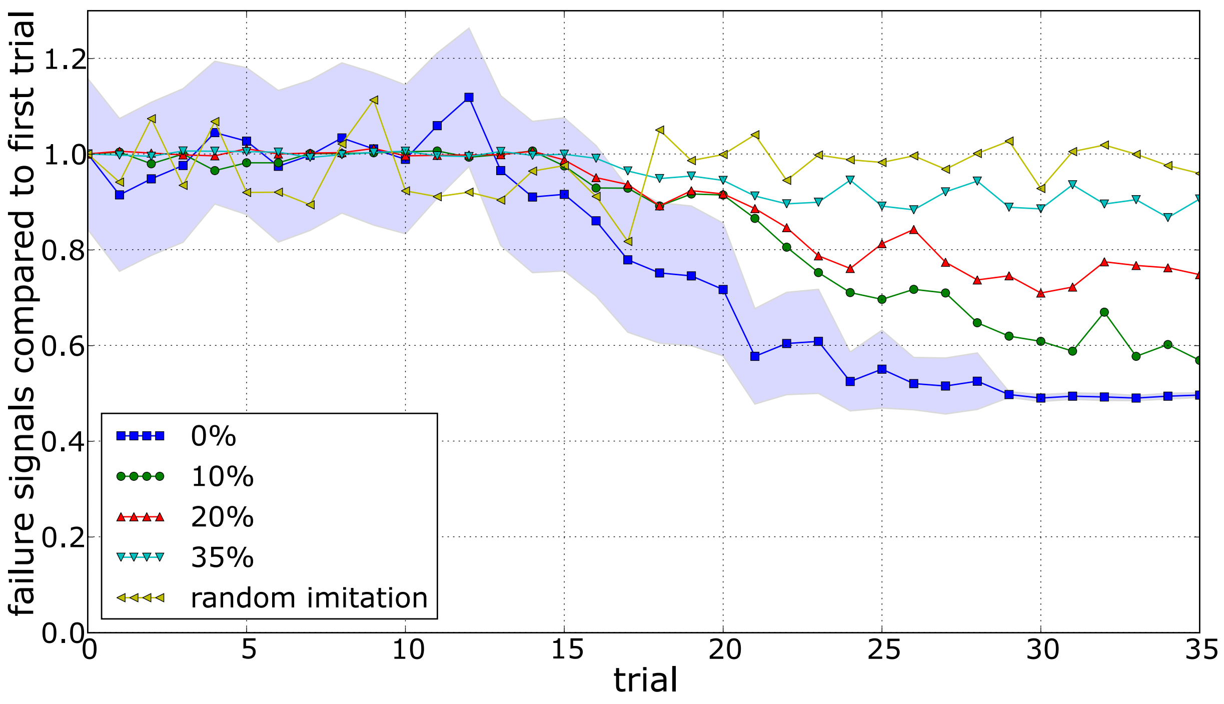


Figure 9.16: Impact of the demonstrator selection algorithm on the failure rates for different fractions of unknown data (with 95% confidence interval for the 0% case)

The graph shows that the algorithm presented in this chapter is able to improve upon that. It does so, however, not before the 15th experiment run. That means that this approach needs a minimal amount of data to show its strengths. Although the failure rate never drops to zero due to the noise in the environment and the realistic simulation scenario, the approach is able to decrease the failure rate by approximately 50% in the case of no unknown data (0% case).

With increasingly more introduced unknown data, the performance degrades as expected. With 35% and more of unknown data it approaches the performance of random imitation. Recall that this approach uses the *Alternating Model Selection EM-algorithm* to learn the affordance networks [76]. At approximately 30% of unknown data, virtually all the instances in the affordance dataset are incomplete.

In summary, the demonstrator selection algorithm is able to improve the imitation performance significantly by choosing demonstrator robots that will most likely provide useful new behavior to the imitator. The algorithm, however, needs a minimal amount of meaningful data. If this cannot be guaranteed the worst thing to happen is a performance as if it had not been used.

9.7.4 Clustering experiment

In the previous experiments, the objects had similar properties. E. g., all nine objects had nearly similar size and weight. This supports the demonstrator selection algorithm as it increases the quality of the collected affordance data. With more variance in the properties of the objects, also the affordance data becomes more ambiguous. If, e. g., nine additional objects were introduced, all of them big and heavy, the collected affordance data would be contradicting for the most part. The experiment in this section investigates how the approach copes with objects that are more heterogeneous.

For this purpose, the robots cluster the objects according to the properties they can perceive. Subsequently, they construct one affordance network for each cluster individually. The applied affordance network metric finally has to be extended so that it incorporates all the individual affordance network distances.

9.7.4.1 Scenario

The experimental setup consists of 18 objects, which can be divided into two groups depending on their perceivable properties. Within each group, the variance of the object properties is low. The robots, however, are not given the number of clusters. They determine this by means of the elbow criteria [27], which can then be used together with the applied agglomerative hierarchical clustering method [124, 153]. It forms bottom-up the most similar pairs of objects to clusters, and then the most similar clusters into bigger clusters, and so on, level for level. At each level, the sum of the clusters' mean square error of the object distances indicates the quality of the clustering. The elbow criteria states that it is advisable to stop the agglomeration process if the error increases steeply from one level to the next.

9.7.4.2 Procedure

The experiment proceeds similar to the previous one, except that the imitator clusters the objects perceived so far prior to the affordance data collection and affordance network creation. It then creates the affordance networks for each robot in each cluster individually. Recall that the whole procedure is performed from the view of one robot. At no point in the experiment, the robots share any data. Therefore, when the imitating robot creates the according affordance networks of the different robots based on its observations, the clustering stays the same.

Assume that the objects have been grouped into n clusters of the set $\{K_1, \dots, K_n\}$. In this case, the imitating robot $\mathcal{R}_m$ creates n affordance networks for each robot. During exploration, $\mathcal{R}_m$ collects the affordances observed at robot $\mathcal{R}_i$ into the n sets $\mathcal{T}^m_{i,1}, \dots, \mathcal{T}^m_{i,n}$ (cf. Eq. (9.5) in Sec. 9.5). Robot $\mathcal{R}_m$ uses them to create the according affordance networks $AN_{i,1}, \dots, AN_{i,n}$. It calculates the cluster-based distance $D^c_{AN}(\mathcal{R}_a, \mathcal{R}_b)$ between two robots $\mathcal{R}_a$ and $\mathcal{R}_b$ by a weighted sum of the individual affordance network distances, where the weight is determined by the amount of affordance data available for the corresponding cluster:

$$D^c_{AN}(\mathcal{R}_a, \mathcal{R}_b) = \sum_{l=1}^{n} \frac{k_l}{k} \cdot D_{AN}(AN_{a,l}, AN_{b,l}) \tag{9.32}$$

Each affordance network $AN_{i,l}$, constructed from data $\mathcal{T}^m_{i,l}$, is weighted by $\frac{k_l}{k}$, where

$$k = \min\left\{ \left(|\mathcal{T}^m_{a,l}| + |\mathcal{T}^m_{b,l}| \right) \mid 1 \le l \le n \right\} \tag{9.33}$$

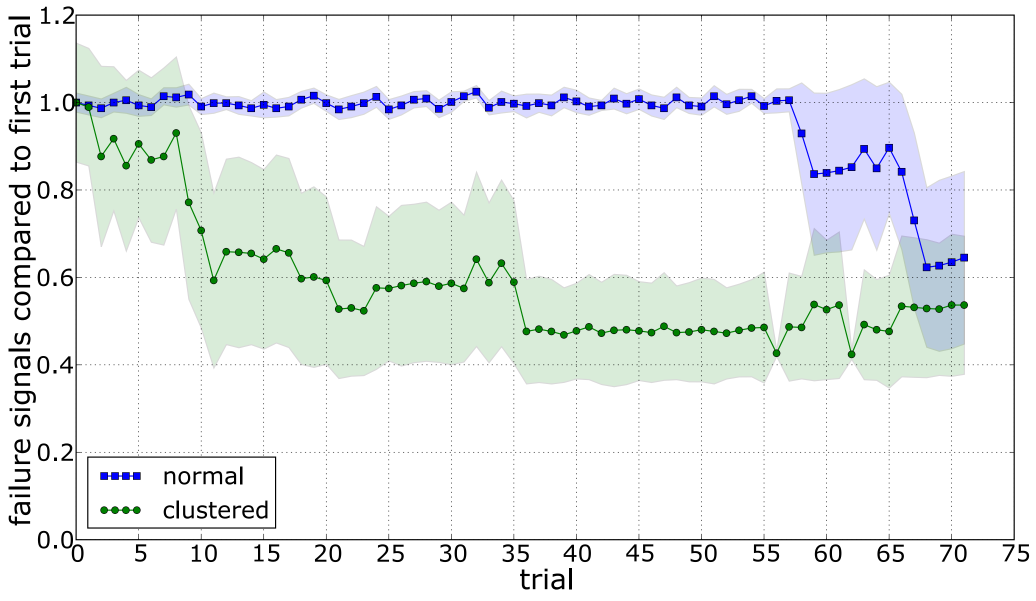


Figure 9.17: Impact of the clustered demonstrator selection algorithm on the failure rates compared to the non-clustered approach

is the minimal amount of affordance data collected by both robots $\mathcal{R}_a$ and $\mathcal{R}_b$. The numerator

$$k_l = \min\left\{|\mathcal{T}^m_{a,l}|, |\mathcal{T}^m_{b,l}|\right\} \tag{9.34}$$

is the minimal amount of objects belonging to cluster K_l explored by both robots. This ensures a higher weight for distance calculations that are based on more data.

The rest of the cluster experiment is done equally to the previous experiment.

9.7.4.3 Results

The graph in Fig. 9.17 shows the performance of the clustering approach ("clustered") and compares it to the not clustered one ("normal") averaged over ten experiments. Without clustering, the failure rates stay the same for over 55 trials and the choice of the best demonstrator is random. This indicates that it needs much more data to construct meaningful affordance networks if the objects show a greater variance in their properties. It takes over 65 trials for the imitator to collect enough data.

In contrast to that, the clustered approach is able to decrease the failure rates very early. After the 35th trial it converges at half of its starting failure rate.

9.8 Conclusion

This chapter solved the problem of determining the best demonstrator to imitate. As it is most reasonable to imitate a robot with similar capabilities, this robot is determined by means of the similarity of the observed affordances compared to the affordances of the imitating robot. Typically, these observations are noisy and ambiguous. To cope with this, the presented demonstrator selection approach constructs affordance networks that are Bayesian networks with affordances as nodes. With a metric that takes into account both the network structure and the affordance dependency parameters, the demonstrator selection approach is able to determine the best robot demonstrator for a given robot.

It does so even without requiring the robots to reveal their inner status of other information. Thus, it is not only robust with respect to the reliability of the communication channel and does not require the same communication protocol. It even works in robot groups where robots are not specifically designed to be used in an imitation scenario.

The approach requires predefined affordances as its basis. If this cannot be requested for a specific scenario, an affordance-learning phase could precede the application of the demonstrator selection approach. Research by Montesano et al. [125, 126, 127] and Detry et al. [66] has shown that this is a viable solution.

The experiments have shown empirically that the demonstrator selection increases the imitation performance while being robust to deterioration of the observation quality. In addition to improving the imitation performance, the approach also saves computing resources otherwise spent at monitoring robots that are too different to imitate.

CHAPTER 10

Summary and outlook

The famous science fiction writer Arthur C. Clarke once formulated his three laws of prediction regarding the state and progress of science [58]:

1. *When a distinguished but elderly scientist states that something is possible, he is almost certainly right. When he states that something is impossible, he is very probably wrong.*
2. *The only way of discovering the limits of the possible is to venture a little way past them into the impossible.*
3. *Any sufficiently advanced technology is indistinguishable from magic.*

Because pushing the technological boundaries forward is such a tough and risky thing, researchers often take inspiration by nature hoping to ease some of those difficulties. When they are trying to copy or emulate processes and mechanisms that have been proven to be successful in nature, one inevitably has to realize and acknowledge the complexity and sophistication governing natural processes. This is also true for imitation and its combination with learning. Whereas imitation observed "in the wild" seems to be so easy and natural, it is hard to accomplish in technical systems. Even more so, if restricting assumptions are omitted. This thesis has presented approaches that are a step forward in this direction, as they allow for sporadic imitation in heterogeneous robot groups. The remainder of this chapter summarizes the contributions of this thesis and discusses possible enhancements.

10.1 Summary

Decent learning capabilities are inevitable for robots in a robot group that has to achieve its goal in a robust manner. Even, if the original task can be programmed completely, unforeseen changes in the environment or of the robots themselves due to wear and tear still require each robot to continuously adapt. This process of initial learning and adapting the learned knowledge can and should be sped up by spreading the learned knowledge among the group members – in form of imitation.

For a robot system to support this combination of learning and imitation, this thesis first developed a robot architecture that supports the required characteristics. It consists of the three layers *motivation*, *strategy*, and *skill*. The overall goal of the robot is specified intuitively in the motivation layer. At runtime, it provides feedback to the robot with respect to the outcome of its last actions. This feedback is used in the strategy layer as a reward information. Based on that, the strategy calculates and updates its current optimal policy. The policy is modeled by a semi-Markov decision process that keeps track of which action is most useful in the respective state. The actions are grounded in the skill layer. This means that for each symbolic action in the strategy layer, the skill layer maintains a low-level representation. This is an approximated function that determines the commands for the actuators. A robot is able to individually learn to achieve the goals specified in the motivation layer on two levels. At first, the skill layer is autonomously able to derive low-level skills that might be useful to achieve the overall goal. And then, there is the strategy layer, which uses these previously learned skills to build its strategy.

If the robot encounters another robot that might serve as a demonstrator, both the skill and strategy layer work together to reconstruct the observed behavior. Inspired by the biological mirror neuron system, the skill layer recognizes behavior in the observation that it could also have performed. Based on this insight, the strategy layer then builds the most likely state-action-trace according to the observation. Since the information is already in a format that is understood by both the strategy layer and skill layer, the recognized behavior in the observation can then be added to the current experience of the strategy layer. This procedure is made possible only by the clear separation of concerns between the strategy and the skill layer.

Another challenge that has been solved in this thesis is the decision regarding the most useful imitatee. This problem aggravates with increasing heterogeneity in the robot group. With the proposed solution, imitation can now be used in completely heterogeneous robot groups. The approach is based on the behavioral difference between two robots. It creates affordance networks, which are Bayesian networks on observed affordances. Once, an imitator has created an affordance network for each robot in a group, it can calculate the difference between itself and all the other robots. By taking the robot with the smallest distance, it has chosen the imitatee that behaves most similar to itself, which increases the probability of successful imitation.

10.2 Contributions

The thesis starts with a survey of imitation in science ranging from biology over psychology to robotics (Chap. 2), which originally connects different fields of research. The remainder delivers the following contributions:

- **An architecture that combines top-down goal specification with bottom-up behavior acquisition (Chap. 3 – Chap. 6).** There are already robot architectures that deploy means to address the inherent complexity in today's robot tasks by combining high-level goal-specification with low-level behavior execution. The approach of this thesis is original in that it has integrated the support for imitation at each layer.

- **An algorithm for non-obtrusive imitation (Chap. 8).** The presented approach only requires the imitatee to emit a signal of its overall well-being, which is used to approximate the observation's overall success. Everything else, the correspondence problem, the "how", "what", "when", and "whom" is handled autonomously by the approach itself. This increases the robot's own autonomy and robustness dramatically.

- **An algorithm to autonomously determine the best imitatee (Chap. 9).** The "how" is handled in a novel manner, as it only requires a set of affordances to be specified, based on which the imitator is then able to calculate the most similar imitatee of potential robots. This advances the state of the art, in that it poses no further requirements to the robots in the robot group.

The contributions have been published in the following journals and conference proceedings:

- *International Journal On Advances in Intelligent Systems* 2009
  [21]

- **IROS:** *IEEE/RSJ International Conference on Intelligent Robots and Systems* 2008
  [18]

- **ADPRL:** *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning* 2009
  [20]

- **BICC:** *IFIP Conference on Biologically Inspired Cooperative Computing* 2006, 2008
  [13, 17, 1]

- **ICAS:** *International Conference on Autonomic and Autonomous Systems* 2007, 2008, 2009
  [9, 16, 19]

- **SEAMS**: *IEEE/ACM ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* 2008
  [10]

A complete list of publications can be found in the appendix.

10.3 Outlook

Since the presented techniques and algorithms are qualitatively new approaches, there is naturally room for more evaluation experience in further areas. It will be interesting to see how well the architecture behaves in situations that are more noisy and uncertain or that involve more objects and robots. Besides the extension of evaluation experience, the following areas can be improved as described in the remainder of this section.

Regarding the architecture, some of the parameters that control the strategy and skill layer are manually predefined and thus have to be found empirically. Further investigation is needed to examine how those can be determined dynamically. This would directly improve the robustness and autonomy of the overall system.

In the presented imitation approach, each robot assumes that the other robots are steered by the same value system, which is the behavior of the motivation layer. I. e., they don't assume the same implementation, but that all the robots agree upon what is beneficial and what is not in terms of the rewards. To increase the applicability of the robots even more, some kind of compatibility check of the motivation system is needed. If robots are able to detect, which robots possess quite different goals, it would rule them out as potential imitatees. Needless imitation attempts would then be prevented.

The quality of the imitation approach could be improved further by implementing joint attention methods [49]. Joint attention is the capability of attending to the same object, to which another robot is looking [68]. This would limit the object to track to those the demonstrator is tracking.

The imitatee selection approach can be improved by reasoning in the imitatee selection step whether it is wise to imitate at all. The current approach selects the most similar robot. If all other robots are dissimilar, the approach, nevertheless, chooses one to imitate. In this case, the approach could decide that no robot within the visible range is "similar enough". This requires an absolute threshold of behavioral similarity and could be determined online in the optimal case.

The presented architecture can even be used as a basis for qualitatively new multi-robot coordination. Current approaches that already enable robots in groups to coordinate their actions so far require fixed coordination rules [34, 83, 84, 158]. This limits the deployment of robot groups to predefined and fixed scenarios where only those robots can benefit that are enabled with the predefined rules, which are needed for coordination. In contrast, coordination based on the presented architecture's capability to recognize complex behavior in observations could overcome this restriction. Based on the continuously recognized complex behavior, each robot could build teammate models of the other robots. These models could then guide the robot to coordinate the actions with the other robots even if they were provided with incompatible coordination means or not specifically designed for coordination at all.

APPENDIX A

Notation

$\boldsymbol{\mathcal{R}}$	set of all robots $\mathcal{R}_i$ in the group
$\boldsymbol{O}$	set of objects in the environment
$\boldsymbol{\mu}$	motivation vector
μ_i	motivation for goal i
μ_i^θ	motivation threshold defining when goal i is satisfied
μ_i^p	priority for goal i
$\mathbb{S}$	state space
$\mathbb{A}$	action space
$V(s)$	value or expected accumulated return of state s
$Q(s,a)$	utility of performing action a in state s
$T(s,a,s')$	probability of arriving at s' when executing action a in state s
$R(s,a)$	expected reward for executing action a in state s
$\pi(s)$	policy that assigns an action so state s
f_{ext}	extraction function
f_c	computes the error associated to a couple of real values
f_e	computes the error of the perception S
f_p	progress function
m	model prediction function
$\mathcal{I}$	raw perception (input)
$\mathcal{I}_{m,s,a}$	preprocessed perception for the motivation, strategy, and skill layer, respectively
$\mathcal{O}$	raw action (output)
$\mathcal{V}$	value of a hidden state in a Viterbi path
$\boldsymbol{\Lambda}$	set of detectable affordances Λ_i
$\boldsymbol{A}$	set of random variables representing affordances in $\boldsymbol{\Lambda}$

APPENDIX B

Algorithms

Algorithm 1 Splitting a reward rate sequence

Input: ρ_{t-n}^{t}: the last n received reward rates

Output: Two reward rate sequences that satisfy Eq. (5.28) or “None”

```
    // Calculate the boundaries the $n$ histogram bins and the number of reward rates that fall into each of them
 1: $h_{bin}, h_{count} \leftarrow$ histogram($\rho_{t-n}^{t}$, bins=$n$)
 2: $i_{max} \leftarrow \arg\max_i h_{count}^{i}$
 3: $bin_{max} \leftarrow h_{bin}^{i_{max}}$
 4: $change \leftarrow \{i \mid (\rho_i \in bin_{max}) \neq (\rho_{i-1} \in bin_{max})\ \forall i \in \{1, \dots, n-1\}\}$
 5: $\rho_{low} \leftarrow$ reward rate sequence without a change and $Var(\rho_{low}) \approx 0$
 6: if $|change| > 1 \vee Var(\rho_{t-n}^{t}) < \theta_V \vee |\rho_{low}| < \theta_l$ then
 7:     return None
 8: else
 9:     $k \in change$
10:     return $\rho_{t-n}^{t-k-1}, \rho_{t-k}^{t}$
```

Algorithm 2 RECOGNIZE: Recognize familiar behavior

Input: O_1^N: observation episode stream $\langle o_1, \dots, o_N \rangle$; $\mathbb{S}$ and $T(s', a, s)$: state space and transition probabilities of the imitator's strategy (SMDP)

Output: Recognized most likely state transitions with observed transition actions

```
 1: Transform $O_1^N$ into subjective observations $\rightarrow o_1^N$ (Sec. 8.3)
 2: $\Gamma \leftarrow \varnothing$        // collects *understood* $\langle s', a, s \rangle$ triples
 3: $\mathcal{V}(s, 1) \leftarrow \max_{s_1} P(o_1 \mid s_1 = s) \ \forall \ s \in \mathbb{S}$
 4: $t_{start}^{rec} \leftarrow 1$
 5: $t_{end}^{rec} \leftarrow N$
 6: $t \leftarrow t_{start}^{rec} + 1$
 7: while $t < |N|$ do
 8:     if $\max_a P_a(o_t \mid o_{t-1}) \geq \theta_{obs}$ then
 9:         for $s \in \mathbb{S}$ do
10:             $a_{t-1} \leftarrow \arg\max_a P_a(o_t \mid o_{t-1})$
11:             $\mathcal{V}(s, t) \leftarrow \max_a P_a(o_t \mid s_t = s) \max_{s'} [T(s', a_{t-1}, s)\mathcal{V}(s', t-1)]$
12:             $\varphi(s, t) \leftarrow \arg\max_{s'} [T(s', a_{t-1}, s)\mathcal{V}(s', t-1)]$
13:     else
14:         $t_{end}^{rec} \leftarrow t$
15:         $\Gamma \leftarrow \Gamma \cup \text{RETRIEVE}(\varphi, t_{start}^{rec}, t_{end}^{rec})$
16:         while $\max_a P_a(o_t \mid o_{t-1}) < \theta_{obs}$ and $t < |N| - 1$ do
17:             $t \leftarrow t + 1$
18:         if $\max_a P_a(o_t \mid o_{t-1}) \geq \theta_{obs}$ then
19:             $t_{start}^{rec} \leftarrow t$
20:             $t_{end}^{rec} \leftarrow N$
21:             $\mathcal{V}(s, t) \leftarrow \max_{s_t} P(o_t \mid s_t) \ \forall \ s \in \mathbb{S}$
22:     $t \leftarrow t + 1$
23: if $t_{start}^{rec} < t_{end}^{rec}$ then
24:     $\Gamma \leftarrow \Gamma \cup \text{RETRIEVE}(\varphi, t_{start}^{rec}, t_{end}^{rec})$
25: return $\Gamma$
```

Algorithm 3 RETRIEVE: Calculate the most likely behavior sequence out of φ

Input: o_1^N, φ, and $\mathbb{S}$ from Alg 2; recognition window $[t_{start}^{rec}, t_{end}^{rec}]$ for o_1^N

Output: The recognized most likely state transitions with observed transition actions for the specified time window

```
 1: $\Gamma_{temp} \leftarrow \varnothing$
 2: $s_{end}^{best} \leftarrow \arg\max_{s \in \mathbb{S}} \mathcal{V}(s, t_{end}^{rec})$
 3: $t \leftarrow t_{end}^{rec}$
 4: while $t \geq t_{start}^{rec}$ do
 5:     if $\varphi(s_{end}^{best}, t) \neq s_{end}^{best} \lor t = t_{start}^{rec}$ then
 6:         $s_{start}^{best} \leftarrow \varphi(s_{end}^{best}, t)$
 7:         $a_{ml} \leftarrow \arg\max_a P_a(s_{end}^{best} \mid s_{start}^{best})$
 8:         $\Gamma_{temp} \leftarrow \Gamma_{temp} \cup \big((t, o_t, s_{start}^{best}), a_{ml}, (t_{end}^{rec}, o_{t_{end}^{rec}}, s_{end}^{best})\big)$
 9:         $t_{end}^{rec} \leftarrow t$
10:         $s_{end}^{best} \leftarrow s_{start}^{best}$
11:     $t \leftarrow t - 1$
12: return $\Gamma_{temp}$
```

List of Figures

List of Tables

Own publications

[1] Raphael Golombek, Willi Richert, Bernd Kleinjohann, and Philipp Adelt. Measurement of robot similarity to determine the best demonstrator for imitation in a group of heterogeneous robots. In *IFIP Conference on Biologically Inspired Cooperative Computing – BICC*. Milano, Italy, 2008.

[2] Bernd Kleinjohann, Lisa Kleinjohann, Willi Richert, and Claudius Stern. Integrating autonomous behavior and team coordination into an embedded architecture. In Pedro U. Lima, editor, *Robot Soccer*, chapter "Integrating autonomous behaviour and team coordination into an embedded architecture", pages 253–280. Pro Literature Verlag / ARS, December 2007.

[3] Markus Koch, Robert Beckebans, Jürgen Schrage, and Willi Richert. Real-time measurement, visualization and analysis of movements by fiber optical sensory applied to robotics. In *47th International IEEE Conference on Instrumentation, Control and Information Technology (SICE 2008)*. Tokyo, Japan, August 2008.

[4] Markus Koch, Willi Richert, and Alexander Saskevic. A self-optimization approach for hybrid planning and socially inspired agents. In *Second NASA GSFC/IEEE Workshop on Radical Agent Concepts*. NASA Goddard Space Flight Center Visitor's Center Greenbelt, MD, USA, 2005.

[5] Markus Koch, Jürgen Schrage, and Willi Richert. Optic-tactile robotics and medical applications. In *International IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM 2008)*. Xi'an, China, July 2008.

[6] Rafael Radkowski, Willi Richert, Henning Zabel, and Philipp Adelt. Augmented reality-based behavior-analysis of autonomous robotic soccers. In *IADIS International Conference of Applied Computation*. Algarve (Portugal), 2008.

[7] Willi Richert and Bernd Kleinjohann. Self–organization at the lowest level: Proactively learning skills in autonomous systems. In *GI 2006 – Organic Computing Workshop*, GI–Edition Lecture Notes in Informatics (LNI), 2006.

[8] Willi Richert and Bernd Kleinjohann. A robust skill learning framework for autonomous mobile robots. In *Proceedings of the 4th International Symposium on Autonomous Minirobots for Research and Edutainment (AMiRE 2007)*, volume 216. HNI-Verlagsschriftenreihe, 2007.

[9] Willi Richert and Bernd Kleinjohann. Towards robust layered learning. In *International Conference on Autonomic and Autonomous Systems (ICAS'07)*. IEEE Computer Society, June 2007.

[10] Willi Richert and Bernd Kleinjohann. Adaptivity at every layer: a modular approach for evolving societies of learning autonomous systems. In *Proceedings of IEEE/ACM ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'08)*, pages 113–120. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-037-1.

[11] Willi Richert, Bernd Kleinjohann, and Lisa Kleinjohann. Evolving agent societies through imitation controlled by artificial emotions. In De-Shuang Huang, Xiao-Ping Zhang, and Guang-Bin Huang, editors, *Advances in Intelligent Computing, International Conference on Intelligent Computing, ICIC 2005, Hefei, China, August 23-26, 2005, Proceedings, Part I*, volume 3644 of *Lecture Notes in Computer Science*, pages 1004–1013. Springer, 2005. ISBN 3-540-28226-2.

[12] Willi Richert, Bernd Kleinjohann, and Lisa Kleinjohann. Learning action sequences through imitation in behavior based architectures. In *Systems Aspects in Organic and Pervasive Computing – ARCS 2005*, number 3432 in LNCS, pages 93–107. Springer-Verlag Berlin, march 2005.

[13] Willi Richert, Bernd Kleinjohann, and Lisa Kleinjohann. Trading off impact and mutation of knowledge by cooperatively learning robots. In *IFIP Conference on Biologically Inspired Cooperative Computing – BICC*, 2006.

[14] Willi Richert, Bernd Kleinjohann, Markus Koch, Alexander Bruder, Stefan Rose, and Philipp Adelt. The Paderkicker Team: Autonomy in realtime environments. In *Proceedings of the Working Conference on Distributed and Parallel Embedded Systems (DIPES)*, 2006.

[15] Willi Richert, Bernd Kleinjohann, and Alexander Murmann. Towards robust skill learning with prediction guided autonomy in unknown environments. In Yuichi Motai and Bernhard Sick, editors, *IEEE Mountain Workshop on Adaptive and Learning Systems*, July 2006.

[16] Willi Richert, Olaf Lüke, Bastian Nordmeyer, and Bernd Kleinjohann. Increasing the autonomy of mobile robots by on-line learning simultaneously at different levels of abstraction. In *International Conference on Autonomic and Autonomous Systems (ICAS'08)*. IEEE Computer Society, March 2008. **Best Paper Award**.

[17] Willi Richert, Oliver Niehörster, and Florian Klompmaker. Guiding exploration by combining individual learning and imitation in societies of autonomous robots. In *IFIP Conference on Biologically Inspired Cooperative Computing – BICC*. Milano, Italy, 2008.

[18] Willi Richert, Oliver Niehörster, and Markus Koch. Layered understanding for sporadic imitation in a multi-robot scenario. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'08)*, pages 1287–1292. Nice, France, 2008.

[19] Willi Richert, Ulrich Scheller, Markus Koch, Bernd Kleinjohann, and Claudius Stern. Increasing the autonomy of mobile robots by imitation in multi-robot scenarios. In *International Conference on Autonomic and Autonomous Systems (ICAS'09)*, 2009.

[20] Willi Richert, Ulrich Scheller, Markus Koch, Bernd Kleinjohann, and Claudius Stern. Integrating sporadic imitation in reinforcement learning robots. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL'09)*, 2009.

[21] Willi Richert and Riccardo Tornese. ESLAS – a robust layered learning framework. *International Journal On Advances in Intelligent Systems*, 2(1):241–253, May 2009.

[22] Claudius Stern, Philipp Adelt, Willi Richert, and Bernd Kleinjohann. Hierarchically distributing embedded systems for improved autonomy. In *Proceedings of the Working Conference on Distributed and Parallel Embedded Systems (DIPES)*, 2008.

Bibliography

[23] Sick sensor inteligence. 2008. URL `http://www.sick.com/at/produkte/produktkataloge/auto/lasermesssystemeindoor/de.html`.

[24] Ranjan Acharyya. *A New Approach for Blind Source Separation of Convolutive Sources*. VDM Verlag, 2008.

[25] ActivMedia. URL for the Pioneer robot: http://www.activrobots.com, 2003.

[26] Aris Alissandrakis, Chrystopher L. Nehaniv, Kerstin Dautenhahn, and Hatfield Hefts All Ab. Synchrony and perception in robotic imitation across embodiments. In *Proc. IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA'03)*, pages 923–930, 2003.

[27] Ethem Alpaydin. *Introduction To Machine Learning*. MIT Press, 2004. ISBN 0262012111.

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

[29] T. Asfour, P. Azad, F. Gyarfas, and R. Dillmann. Imitation Learning of Dual-Arm Manipulation Tasks in Humanoid Robots. *International Journal of Humanoid Robotics (IJHR)*, 2008.

[30] Robert Aunger. *The Electric Meme*, pages 268–275. Free Press, 2002.

[31] Pedram Azad, Tamim Asfour, and Rüdiger Dillmann. Toward an unified representation for imitation of human motion on humanoids. In *IEEE International Conference on Robotics and Automation (ICRA'07)*, pages 2558–2563, 2007.

[32] Tucker R. Balch. *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998.

[33] Tucker R. Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8(3):209–238, 2000.

[34] G. Baldassarre, V. Trianni, M. Bonani, F. Mondada, M. Marco, and S. Nolfi. Self-Organized Coordinated Motion in Groups of Physically Connected Robots. *IEEE Transactions on Systems, Man and Cybernetics*, 37(1):224, 2007.

[35] J.M. Baldwin. *Development and Evolution*. New York, Macmillan, 1902.

[36] L.F. Baptista and L. Petrinovich. Social interaction, sensitive phases, and the song template hypothesis in the white-crowned sparrow. *Animal Behaviour*, 32:172–181, 1984.

[37] Jeffrey S. Beis and David G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 00, 1997.

[38] F. Bellagamba and M. Tomasello. Re-enacting intended acts: Comparing 12- and 18-month-olds. *Infant Behavior and Development*, 22:277–282, 1999.

[39] R.E. Bellman. *Dynamic Programming*. Courier Dover Publications, 2003.

[40] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[41] D.C. Bentivegna. *Learning from Observation Using Primitives*. PhD thesis, Georgia Institute of Technology, 2004.

[42] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[43] A. Billard, Y. Epars, S. Calinon, G. Cheng, and S. Schaal. Discovering optimal imitation strategies. *Robotics and Autonomous Systems*, 47(2–3):69–77, 2004.

[44] Blackmore, S. *The Meme Machine*. Oxford University Press, 1999. ISBN 0-19-286212-X.

[45] A. Bonarini, A. Lazaric, and M. Restelli. Reinforcement learning in complex environments through multiple adaptive partitions. In *Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence (AI*IA)*, pages 531–542, 2007.

[46] Josh Bongard and Hod Lipson. Automatic synthesis of multiple internal models through active exploration. In *AAAI Fall Symposium: From Reactive to Anticipatory Cognitive Embodied Systems, 2005*, November 2005.

[47] Joost Broekens. Emotion and reinforcement: Affective facial expressions facilitate robot learning. In *Artifical Intelligence for Human Computing*, pages 113–132, 2007.

[48] E. Burnstein, E. Stotland, and A. Zander. Similarity to a model and self-evaluation. *Journal of Abnormal and Social Psychology*, 62:257–264, 1961.

[49] G. Butterworth and N. Jarett. What minds have in common is space: Spatial mechanisms serving joint visual attention in infancy. *British Journal of Developmental Psychology*, 9: 55–72, 1991.

[50] Richard W. Byrne. Animal imitation. *Current Biology*, 19:R111–R114, 2009.

[51] S. Calinon. *Robot Programming by Demonstration: A Probabilistic Approach*. EPFL/CRC Press, 2009.

[52] S. Calinon and A. Billard. Learning of gestures by imitation in a humanoid robot. In *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions*, pages 153–177. Cambridge University Press, K. Dautenhahn and C.L. Nehaniv edition, 2007.

[53] S. Calinon and A. Billard. Statistical learning by imitation of competing constraints in joint space and task space. *Advanced Robotics*, 2009.

[54] Josep Call and Malinda Carpenter. Three sources of information in social learning. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in animals and artifacts*, pages 211–228. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7.

[55] Josep Call and M. Tomasello. The social learning of tool use by orangutans (pongo pygmaeus). *Human Evolution*, 9:297–313, 1994.

[56] Josep Call and M. Tomasello. The use of social information in the problem-solving of orangutans (pongo pygmaeus) and human children (homo sapiens). *Journal of Comparative Psychology*, 109(3):308–320, 1995.

[57] Y. Uny Cao, Alex S. Fukunaga, and Andrew B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1):7–23, March 1997.

[58] Arthur C Clarke. *Profiles of the Future*. Littlehampton Book Services Ltd, 1962.

[59] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Learning for control from multiple demonstrations. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008)*, pages 144–151, 2008.

[60] Adam Coates, Pieter Abbeel, and Andrew Y. Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105, 2009.

[61] Charles N. Cofer. *Motivation and emotion*. Scott, Foresman and Co., Glenview, Ill., 1972. 176 S. pp.

[62] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, 1967.

[63] J.P. Crutchfield. Information and its metric. In *Nonlinear Structures in Physical Systems – Pattern Formation, Chaos and Waves*, pages 119–130. Springer Verlag, 1990.

[64] K. Dautenhahn and C. Nehaniv. *Imitation in Animals and Artifacts*, chapter "An agent-based perspective on imitation". MIT Press, 2002.

[65] R. Dawkins. *The Selfish Gene*. Oxford University Press, Oxford, 1976.

[66] R. Detry, E. Baseski, M. Popovic, Y. Touati, N. Krüger, O. Kroemer, J. Peters, and J. Piater. Learning object-specific grasp affordance densities. In *Proceedings of the 8th IEEE International Conference on Development and Learning (ICDL 2009)*, 2009.

[67] Peter J. Dickinson, Horst Bunke, Arek Dadej, and Miro Kraetzl. On graphs with unique node labels. In *Graph Based Representations in Pattern Recognition*, volume 2726, pages 409–437. Springer Berlin, Heidelberg, DE, 2003. ISBN 978-3-540-40452-1.

[68] M.W. Doniec, G. Sun, and B. Scassellati. Active Learning of Joint Attention. *IEEE-RAS International Conference on Humanoid Robotics (Humanoids 2006)*, 2006.

[69] M. Dorigo. Swarm-bot: An experiment in swarm robotics. In P. Arabshahi and A. Martinoli, editors, *Proceedings of SIS 2005 – 2005 IEEE Swarm Intelligence Symposium*, pages 192–200. IEEE Press, Piscataway, NJ, 2005.

[70] Gerry Dozier. Evolving robot behavior via interactive evolutionary computation: from real-world to simulation. In *Proceedings of the 2001 ACM symposium on Applied computing (SAC '01)*, pages 340–344. ACM, New York, NY, USA, 2001.

[71] K. Driessens and S. Džeroski. Integrating Guidance into Relational Reinforcement Learning. *Machine Learning*, 57(3):271–304, 2004.

[72] G. Evangelidis, D. Lomet, and B. Salzberg. The hB ˆΠ-tree: a multi-attribute index supporting concurrency, recovery and node consolidation. *The VLDB Journal The International Journal on Very Large Data Bases*, 6(1):1–25, 1997.

[73] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002. ISBN 0130851981.

[74] Jerome H. Friedman, F. Baskett, and L.J. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, C-24(10):1000–1006, 1975.

[75] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[76] Nir Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. 14th International Conference on Machine Learning*, pages 125–133. Morgan Kaufmann, 1997.

[77] Lena Mariann Garder and Mats Erling Hovin. Robot gaits evolved by combining genetic algorithms and binary hill climbing. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation (GECCO '06)*, pages 1165–1170. ACM, New York, NY, USA, 2006.

[78] Yiannis Gatsoulis, George Maistros, Yuval Marom, and Gillian Hayes. Learning to forage through imitation. In *Proceedings of the Second IASTED International Conference on Artificial Intelligence and Applications (AIA2002)*, pages 485–491, September 2002.

[79] Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *Proceedings of the International Conference on Advanced Robotics*, pages 317–323. Coimbra, Portugal, Jul 2003.

[80] James J. Gibson. *The Senses Considered as Perceptual Systems*. Houghton-Mifflin Company, Boston, 1966.

[81] J.J. Gibson. The theory of affordances. In R. Shaw and J. Brandsford, editors, *Perceiving, Acting, and Knowing: Toward and Ecological Psychology*, pages 62–82. Erlbaum, Hillsdale, NJ, 1977.

[82] Raphael Golombek. Imitationssteuerung durch Messen von Ähnlichkeiten zwischen Roboterverhalten. Diploma thesis, University of Paderborn, 2008.

[83] Roderich Gross and Marco Dorigo. Evolution of solitary and group transport behaviors for autonomous robots capable of self-assembling. *Adaptive Behavior*, 16(5):285–305, 2008.

[84] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.

[85] W.D. Hamilton. Geometry of the selfish herd. *Journal of Theoretical Biology*, 31:295–311, 1971.

[86] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.

[87] R.I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[88] Thomas Hawkins. *Lebesgue's Theory of Integration: Its Origins and Development*. Chelsea, New York, second edition, 1979.

[89] M. Hersch, F. Guenter, S. Calinon, and A. Billard. Dynamical system modulation for robot learning via kinesthetic demonstrations. *IEEE Transaction on Robotics*, 24(6):1463–1467, 2008.

[90] C.M. Heyes. Reflections on self-recognition in primates. *Animal Behavior*, 47:909–919, 1994.

[91] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[92] J.A. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *International conference on robotics and automation (ICRA)*, 2002.

[93] T. Inamura, Y. Nakamura, H. Ezaki, and I. Toshima. Imitation and primitive symbol acquisition of humanoids by the integrated mimesis loop. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, 2001.

[94] T. Inamura, I. Toshima, Y. Nakamura, and J. Saitama. Acquiring Motion Elements for Bidirectional Computation of Motion Recognition and Generation. *Experimental Robotics VIII*, 2003.

[95] Matthew Johnson and Yiennis Demiris. Abstraction in recognition to solve the correspondence problem for robot imitation. In *Towards Autonomous Robotic System (TAROS)*, pages 63–70, 2004.

[96] W. Kadous, Claude Sammut, and R. Sheh. Autonomous traversal of rough terrain using behavioural cloning. In *The 3rd International Conference on Autonomous Robots and Agents*, 2006.

[97] C. Keysers, B. Wicker, V. Gazzola, J.-L. Anton, L. Fogassi, and V. Gallese. A touching sight: SII/PV activation during the observation and experience of touch. *Neuron*, 42(22):1–20, 2004.

[98] A.J. Kinnaman. Mental life of two macacus rhesus monkeys in captivity. *The American Journal of Psychology*, 13(2):173–218, 1902.

[99] Paul R. Kleinginna and Anne M. Kleinginna. A categorized list of motivation definitions, with a suggestion for a consensual definition. *Motivation and Emotion*, 5(3):263–291, September 1981.

[100] M.J. Kochenderfer. *Adaptive Modelling and Planning for Learning Intelligent Behaviour*. PhD thesis, School of Informatics, University of Edinburgh, 2006.

[101] M.J. Kochenderfer. Adaptive Modelling and Planning for Learning Intelligent Behaviour. 2006.

[102] M.J. Kochenderfer and G. Hayes. Modeling and planning in large state and action spaces. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.

[103] Sven Koenig and Reid G. Simmons. Unsupervised learning of probabilistic models for robot navigation. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2301–2308, 1996.

[104] T. Kohonen. *Self-Organizing Maps*. New York: Springer-Verlag, 3rd edition, 2001.

[105] Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM J. Control Optim.*, 42(4):1143–1166, 2003.

[106] Stan A. Kuczaj and Deirdre B. Yeater. Dolphin imitation: Who, what, when, and why? *Aquatic Mammals*, 32:413–422(10), December 2006.

[107] Thomas H. Labella, Marco Dorigo, and Jean-Louis Deneubourg. Division of labor in a group of robots inspired by ants' foraging behavior. *ACM Trans. Auton. Adapt. Syst.*, 1(1): 4–25, 2006.

[108] A. Laud and G. DeJong. The influence of reward on the speed of reinforcement learning: An analysis of shaping. *Proceedings of the Twentieth International Conference (ICML 2003)*, pages 21–24, 2003.

[109] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 833–840. MIT Press, Cambridge, MA, 2008.

[110] Dongheui Lee and Yoshihiko Nakamura. Probabilistic model of whole-body motion imitation from partial observations. In *Proceedings of the 12th International Conference on Advanced Robotics (ICAR'05)*, pages 337–343. Seattle, July 2005.

[111] Yuming Liang and Lihong Xu. Mobile robot global path planning using hybrid modified simulated annealing optimization algorithm. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation (GEC '09)*, pages 309–314. ACM, New York, NY, USA, 2009.

[112] L.J. Lin. Programming robots using reinforcement learning and teaching. *Proceedings of AAAI*, 91:781–786, 1991.

[113] M.L. Littman, S. Singh, and T. Jaakkola. Szepesvári: Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning Journal*, pages 287–308, 1998.

[114] Wenguo Liu. *Design and modelling of Adaptive Foraging in Swarm Robotic Systems*. PhD thesis, Univeristy of the West of England, Bristol, UK, 2008.

[115] David B. Lomet and Betty Salzberg. The hb-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, 1990.

[116] Christopher Lörken and Joachim Hertzberg. Grounding planning operators by affordances. In *Proceedings of the 2008 International Conference on Cognitive Systems*, 2008.

[117] Bhaskara Marthi. Automatic shaping and decomposition of reward functions. In *Proceedings of the 24th international conference on Machine learning (ICML 2007)*, pages 601–608. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-793-3.

[118] Abraham Maslow. A theory of human motivation. *Psychological Review*, 50:370–396, 1943.

[119] Maja J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4(1):73–83, 1997.

[120] M.J. Mataric. Reward functions for accelerated learning. *Proceedings of the Eleventh International Conference on Machine Learning*, 189, 1994.

[121] F.S. Melo and M.I. Ribeiro. Reinforcement learning with function approximation for cooperative navigation tasks. In *IEEE International Conference onRobotics and Automation (ICRA 2008)*, pages 3321–3327, 2008.

[122] A.N. Meltzoff. Understanding the intentions of others: Re-enactment of intended acts by 18-month-old children. *Developmental Psychology*, 31:838–850, 1995.

[123] Andrew N. Meltzoff and M. Keith Moore. Imitation of facial and manual gestures by human neonates. *Science*, 198(4312):75–78, October 1977.

[124] Tom M. Mitchell. *Machine Learning*. The McGraw-Hill Companies Inc., 1 edition, 1997. ISBN 0070428077.

[125] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Affordances, development and imitation. In *IEEE 6th International Conference on Development and Learning (ICDL)*, pages 270–275, 2007.

[126] Luis Montesano, M. Lopes, A. Bernardino, and Jose Santos-Victor. Modeling affordances using bayesian networks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4102–4107, 2007.

[127] Luis Montesano, Manuel Lopes, Alexandre Bernardino, and José Santos-Victor. Learning object affordances: From sensory-motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, 2008.

[128] A.W. Moore and C.G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[129] K. Murphy. A Brief Introduction to Graphical Models and Bayesian Networks, 1998. URL `http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html`.

[130] R. Olguin Nagell, K. and M. Tomasello. Processes of social learning in the tool use of chimpanzees (pan troglodytes) and human children (homo sapiens). *Journal of Comparative Psychology*, 107:174–186, 1993.

[131] Richard E. Neapolitan. *Learning Bayesian Networks*. Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

[132] C.L. Nehaniv and K. Dautenhaun. *Imitation in Animals and Artifacts*, chapter 2: "The Correspondance Problem", pages 41–61. MIT Press, 2002.

[133] Y. Nejigane, M. Shimosaka, T. Mori, and T. Sato. Online action recognition with wrapped boosting. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 1389–1395, 2007.

[134] Yu. Nejigane, M. Shimosaka, T. Mori, and T. Sato. Online action recognition with wrapped boosting. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 1389–1395, 2007.

[135] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999)*, pages 278–287. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-612-2.

[136] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, San Francisco, 1998.

[137] Kees C.J. Overbeeke and Stephan S.A.G. Wensveen. From perception to experience, from affordances to irresistibles. In *DPPI '03: Proceedings of the 2003 international conference on Designing pleasurable products and interfaces*, pages 92–97. ACM, New York, NY, USA, 2003. ISBN 1-58113-652-8.

[138] M. Perera, T. Shiratori, S. Kudoh, A. Nakazawa, and K. Ikeuchi. Multilinear analysis for task recognition and person identification. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 1409–1415, 2007.

[139] J. Piaget. *The Child's Construction of Reality*. London: Routledge and Kegan Paul, 1955.

[140] Rosalind W. Picard. *Affective Computing*. The MIT Press, Cambridge, Massachusetts, 1997. ISBN 0-262-16170-2.

[141] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In *Advances in optimization and numerical analysis, Proceedings of the Sixth workshop on Optimization and Numerical Analysis*, pages 51–67. Academic Publishers, Oaxaca,Mexico, 1994.

[142] Steffen Priesterjahn. *Online imitation and adaptation in modern computer games*. PhD thesis, University of Paderborn, 2008.

[143] Steffen Priesterjahn and Markus Eberling. Imitation learning in uncertain environments. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pages 950–960. Springer-Verlag, Berlin, Heidelberg, 2008.

[144] Steffen Priesterjahn and Alexander Weimer. An evolutionary online adaptation method for modern computer games based on imitation. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 344–345. ACM, New York, NY, USA, 2007.

[145] Steffen Priesterjahn, Alexander Weimer, and Markus Eberling. Real-time imitation-based adaptation of gaming behaviour in modern computer games. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1431–1432. ACM, New York, NY, USA, 2008.

[146] Octavian Procopiuc, Pankaj K. Agarwal, Lars Arge, and Jeffrey Scott Vitter. Bkd-tree: A dznamic scalable kd-tree. In *SSTD*, pages 46–65, 2003.

[147] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, April 1994. ISBN 0471619779.

[148] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[149] J. Randløv and P. Alstrøm. Learning to Drive a Bicycle using Reinforcement Learning and Shaping. In J.W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 463–471, 1998.

[150] G. Rizzolatti and L. Craighero. The mirror-neuron system. *Annual Review of Neuroscience*, 27(1):169–192, 2004.

[151] E.T. Rolls. Précis of The brain and emotion. *Behavioral and Brain Sciences*, 23(02):177–191, 2000.

[152] G.J. Romanes. *Animal Intelligence*. London, Kegan Paul Trench, 1882.

[153] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[154] J.D. Salamone. *Behavioral pharmacology of dopamine systems: A new synthesis*. John Wiley and Sons Ltd, 1991. 599–614 pp.

[155] J.D. Salamone. Complex motor and sensorimotor functions of striatal and accumbens dopamine: involvement in instrumental behavior processes. *Psychopharmacology*, 107(2): 160–174, 1992.

[156] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 385–393. Aberdeen: Morgan Kaufmann, 1992.

[157] Joe Saunders, Chrystopher L. Nehaniv, and Kerstin Dautenhahn. Teaching robots by moulding behavior and scaffolding the environment. In *HRI '06: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 118–125. ACM, New York, NY, USA, 2006. ISBN 1-59593-294-1.

[158] Brian Scassellati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12(1):13–24, 2002.

[159] Ulrich Scheller. Lernen komplexer Verhalten in Robotergruppen durch Imitation und Reinforcement-Learning. Diploma thesis, University of Paderborn, 2008.

[160] Alan P. Sexton and Richard Swinbank. Bulk loading the m-tree to enhance query performance. In M. Howard Williams and Lachlan M. MacKinnon, editors, *British National Conference on Databases (BNCOD)*, volume 3112 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2004.

[161] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, 1948.

[162] Q. Shen, J. Saunders, H. Kose-Bagci, and K. Dautenhahn. Acting and interacting like me? A method for identifying similarity and synchronous behaviour between a human and robot. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'08)*. Nice, France, 2008.

[163] Aaron P. Shon, Keith Grochow, and Rajesh P.N. Rao. Robotic imitation from human motion capture using gaussian processes. In *Proceedings of the IEEE/RAS International Conference on Humanoid Robots (Humanoids)*, 2005.

[164] Tania Singer, Ben Seymour, John P. O'Doherty, Klaas E. Stephan, Raymond J. Dolan, and Chris D. Frith. Empathic neural responses are modulated by the perceived fairness of others. *Nature*, January 2006.

[165] Satinder P. Singh, Tommi Jaakkola, and Michael I. Jordan. Learning without state-estimation in partially observable markovian decision processes. In *In Proceedings of the Eleventh International Conference on Machine Learning (ICML 1994)*, pages 284–292, 1994.

[166] W.D. Smart and L.P. Kaelbling. Practical reinforcement learning in continuous spaces. *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 903–910, 2000.

[167] C. Sminchisescu, A. Kanaujia, and D. Metaxas. Conditional models for contextual human motion recognition. In *IEEE Computer Vision and Image Understanding*, volume 104, pages 210–220. Elsevier, 2006.

[168] Russell Smith. Website of ODE (Open Dynamics Engine). http://www.ode.org/, 2008.

[169] K.W. Spence. *Behavior theory and conditioning*. Yale University Press, New Haven, 1956.

[170] Alexander Stoytchev. Toward Learning the Binding Affordances of Objects: A Behavior-Grounded Approach. 2005. Developmental Robotics.

[171] M. Sugisaka, A. Loukianov, F. Xiongfeng, T. Kubik, and K.B. Kubik. Development of an artificial brain for liferobot. *Applied Mathematics and Computation*, 164(2):507–521, 2005.

[172] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, 1998.

[173] RS Sutton and AG Barto. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1): 126–134, 1999.

[174] Y. Takahashi, K. Noma, and M. Asada. Rapid behavior learning in multi-agent environment based on state value estimation of others. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, pages 76–81, 2007.

[175] Yasutake Takahashi, Yoshihiro Tamura, and Minoru Asada. Mutual development of behavior acquisition and recognition based on value system. In *From Animals to Animats 10, 10th International Conference on Simulation of Adaptive Behavior (SAB 2008)*, pages 291–300, 2008.

[176] J.T. Tapp. Activity, reactivity, and the behavior-directing properties of stimuli. In J.T. Tapp, editor, *Reinforcement and Behavior*, pages 148–178. New York: Academic Press, 1969.

[177] Hari Thiruvengada and Ling Rothrock. Affordance-based computational model of driver behavior on highway systems: A colored petri net approach. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 888–893, 2007.

[178] E.L. Thorndike. Animal intelligence: An experimental study of the associative processes in animals. *Psychological Review Monographs*, 2(8), 1898.

[179] W.H. Thorpe. *Learning and Instinct in Animals*. London: Methuen, 1956.

[180] Riccardo Tornese. AMAF – Automatic Modular Action Framework. Master's thesis, Politecnico di Milano, Italy, 2009. To be submitted.

[181] H. van Hasselt and M.A. Wiering. Reinforcement learning in continuous action spaces. In *Approximate Dynamic Programming and Reinforcement Learning (ADPRL'07)*, pages 272–279, April 2007.

[182] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

[183] Daniel Wagner and Dieter Schmalstieg. ARToolKitPlus for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, February 2007.

[184] E.A. Wasserman and L. Castro. Surprise and Change: Variations in the Strength of Present and Absent Cues in Causal Learning. *Learning & Behavior*, 33(2):131–146, 2005.

[185] Alan Watt. *3D Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993. ISBN 0201631865.

[186] Steven D. Whitehead. A complexity analysis of cooperative mechanisms in reinforcement learning. In *Proceedings of the Ninth National Conference on Artificial Intelligence, volume 2. AAAI Press*, pages 607–613, 1991.

[187] A. Whiten and R. Ham. *Advances in the Study of Behavior*, chapter "On the nature and evolution of imitation in the animal kingdom: Reappraisal of a century of research", pages 239–283. New York: Academic Press, 1992.

[188] Bruno Wicker, Christian Keysers, Jane Plailly, Jean-Pierre Royet, Vittorio Gallese, and Giacomo Rizzolatti. Both of us disgusted in my insula: The common neural basis of seeing and feeling disgust. *Neuron*, 40(3):655–664, 2003.

[189] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Proceedings of the Computer Vision and Pattern Recognition (CVPR '92)*, pages 379–385, 1992.

[190] J. Zhang and V.L. Patel. Distributed cognition, representation, and affordance. *Cognition & Pragmatics*, 14(2):333–341, 2006.