

Structural Knowledge Transfer by Spatial Abstraction for Learning Agents

(Structural Knowledge Transfer by Spatial Abstraction)

Lutz Frommberger and Diedrich Wolter
(SFB/TR 8 Spatial Cognition, University of Bremen)

Contact:

Lutz Frommberger
AG Cognitive Systems
University of Bremen
P.O. Box 330 440
28334 Bremen, Germany
Phone: +49-421-218-64 281
Fax: +49-421-218-98 64 281
`lutz@sfbtr8.uni-bremen.de`

Abstract

In this article we investigate the role of abstraction principles for knowledge transfer in agent control learning tasks. We analyze abstraction from a formal point of view and characterize three distinct facets: aspectualization, coarsening, and conceptual classification. The taxonomy we develop allows us to interrelate existing approaches to abstraction, leading to a code of practice for designing knowledge representations that support knowledge transfer. We show that abstraction is inherently task-dependent which requires abstraction to be studied in an action-centered manner. Furthermore, we detail how abstraction principles can be utilized to achieve knowledge transfer in reinforcement learning. We propose the use of so-called structure space aspectualizable knowledge representation that explicate structural properties of the state space and present a posteriori structure space aspectualization as a method to transfer learned policies across different scenarios. Finally, we present a case study that demonstrates knowledge transfer of generally sensible navigation skills from simple simulation to a real-world robotic platform.

Keywords: abstraction, knowledge transfer, reinforcement learning, transfer learning, robot navigation

1 Introduction

An essential quality of a cognitive being is its ability to learn, that is, to gain new knowledge or skills as well as to improve existing knowledge or skills based on experience. Cognitive beings are able to cope with situations they have been previously confronted with as well as they are able to adapt to new situations sensibly. Thus, when designing an artificial agent that shall exhibit cognitive qualities—which we refer to in short as a *cognitive agent*—one central task is to develop computational means that enable the agent to learn. In this paper, we subscribe to one of the most influential paradigms of machine learning, *reinforcement learning* (RL) (Sutton & Barto, 1998). Reinforcement learning is very valuable when the characteristics of the underlying system are not known and/or difficult to describe or when the environment of an acting agent is only partially known or completely unknown.

Agents situated in the real world can perceive a great variety of information. Two situations are unlikely to lead to the same perception even if the situations are very similar. Learning requires the agent to acknowledge important details in a perception, to recognize commonalities across situations, and, most importantly, to disregard irrelevant information. In other words, the ability to learn involves the ability to *abstract*. Abstraction enables us to conceptualize the surrounding world, to build categories, and to derive reactions from these categories that can adapt to different situations. Complex and overly detailed circumstances can be reduced to much simpler concepts and not until then it becomes feasible to deliberate about conclusions to draw and actions to take. Abstract concepts explicate commonalities in different scenes and, thus, can cover new situations.

Abstraction has been shown to be suitable to tackle the poor learning performance of reinforcement learning in large and continuous state spaces as we encounter them approaching real world tasks. Significant improvements have been achieved by applying abstraction both to the agent’s actions (Sutton, Precup, & Singh, 1999; Parr & Russell, 1998; Dietterich, 1998) as well as to the agent’s representation of the state space (Dean & Givan, 1997; Munos & Moore, 1999; Reynolds, 2000). In this work, we address another shortcoming of reinforcement learning: A found solution relates to one particular problem in a specific environment. The question how this solution can be utilized to enable the cognitive agent to adapt a learned behavior to new tasks and/or similar situations is a big challenge in machine learning research. For this kind of knowledge reuse the term *transfer learning* has become popular. We demonstrate the impact of state space abstraction for transfer learning in RL tasks: As abstract concepts summarize commonalities in different situations, they can enable knowledge transfer from one situation to another. This requires that a state space representation is chosen in a way that applies both to the task where knowledge is acquired and the one it is transferred to.

Abstraction comes in various facets, and it is the system designer’s responsibility to take care for the right choice of abstraction. We are convinced that an understanding is best based on a formal framework, because only a consistent formalization of abstraction allows for a thorough investigation of its properties and effects. So the first contribution of this paper is a formal framework of abstraction based on which we can show that two facets of abstraction, namely aspectualization and coarsening, are equally expressive (Section 2). As aspectualization allows for easy access to relevant aspects, we then argue for the use of this abstraction facet. It allows for building what we call structure space aspectualizable state spaces (Section 3), which enable easy extraction of generally sensible behavior that is related to the structure of the problem and applies to different problems. We also present an algorithm to single out generally sensible behavior from a policy and show how this can be used for knowledge reuse when learning new tasks with reinforcement learning (Sec-

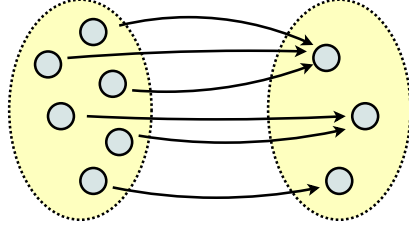


Figure 1: Abstraction is a non-injective mapping: different states collapse to identical entities in the target space.

tion 4). Finally, we report on an experimental validation of transfer learning success where we also demonstrate that we can transfer a robot navigation strategy learned in a dramatically simplified environment to control a mobile robot in the real world (Section 5).

2 A Formal Framework of Abstraction

In their thorough study on abstraction, which they refer to as “schematization”, Klippel, Richter, Barkowsky, and Freksa (2005) state that “there is no consistent approach to model schematization”, because most scientific work on abstraction remains informal or based on examples. In contrast to that, we take a computer scientist’s view on abstraction to provide a formal framework thereof. We distinguish three facets of abstraction, namely *aspectualization*, *coarsening*, and *conceptual classification*. Investigating the relationship of these facets and their implication on agent control processes, principles of state space abstraction for cognitive agents can be concluded.

2.1 Defining Abstraction

When talking about abstraction in the context of information processing and cognitive science, abstraction covers more than only taking away something, because it is not merely intended to reduce the amount of data. Rather, abstraction puts the focus on *relevant* information.

Definition 1. *Abstraction is the process or the result of reducing the information of a given observation in order to achieve a classification that omits information irrelevant for a particular purpose.*

We first concentrate on information reduction. Let us say that all potential values of a knowledge representation are elements of a set \mathcal{S} , the *state space*. Without loss of generality we assume in the following, simply to ease readability, that \mathcal{S} is a Cartesian product of features that all stem from one domain \mathcal{D} , that is, $\mathcal{S} = \mathcal{D}^n$. We call $s = (s_1, \dots, s_n) \in \mathcal{S}$ a *feature vector*, and every s_i is a *feature*. Each feature vector defines a *state*.

Abstraction is a non-injective function $\kappa : \mathcal{S} \rightarrow \mathcal{T}$ mapping the source space \mathcal{S} to a target space \mathcal{T} , that is, at least two states $s, s' \in \mathcal{S}$ are mapped to one $t \in \mathcal{T}$ (see Figure 1). An injective function κ would be no abstraction, as no information reduction is obtained.

Abstraction can have several forms. Various terms have been coined for abstraction principles, distributed over several scientific fields like cognitive science, artificial intelligence, architecture, linguistics, geography, and many more. Among others we find the terms *granularity* (Zadeh, 1979;

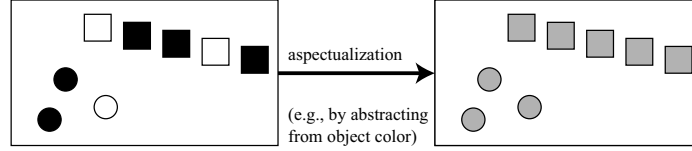


Figure 2: Aspectualization abstracts from certain aspects. In this case, color information is omitted.

Hobbs, 1985; Bittner & Smith, 2001), *generalization* (Mackaness & Chaudhry, 2008), *schematization* (Klippel et al., 2005; Herskovits, 1998), *idealization* (Herskovits, 1998), *selection* (Herskovits, 1998; Stell & Worboys, 1999), *amalgamation* (Stell & Worboys, 1999), or *aspectualization* (Bertel, Freksa, & Vrachliotis, 2004). Unfortunately, some of these terms define overlapping concepts, different terms sometimes have the same meaning, or a single term is used for different concepts. Also, these terms are often not distinguished in an exact manner or only defined by examples.

In this work we study abstraction as part of knowledge representation. The primary concern is representation of spatial knowledge in a way that maintains a perspective as general as possible, allowing for adaptation to other domains.

2.2 Facets of Abstraction

In the following, we characterize three different facets of abstraction: *aspectualization*, *coarsening*, and *conceptual classification*.

2.2.1 Aspectualization

An *aspect* is a piece of information that represents one certain property. For example, if we record the trajectory of a moving robot, we have a spatio-temporal data set denoting at what time the robot visited which place. Time and place are two different aspects of this data set. Aspectualization singles out such aspects (see Figure 2 for illustration).

Definition 2. *Aspectualization is the process or result of explicating certain aspects of an observation by eliminating the others. Formally, it is defined as a function $\kappa : \mathcal{D}^n \rightarrow \mathcal{D}^m$ ($n, m \in \mathbb{N}, n > m$):*

$$\kappa(s_1, s_2, \dots, s_n) = (s_{i_1}, s_{i_2}, \dots, s_{i_m}), \quad i_k \in [1, n], \quad i_k < i_{k+1} \forall k. \quad (1)$$

Aspectualization projects \mathcal{D}^n to \mathcal{D}^m : The dimension of the state space is reduced.

Example 1. *An oriented line segment s in the plane is represented as a point $(x, y) \in \mathbb{R}^2$, a direction $\theta \in [0, 2\pi]$, and a length $l \in \mathbb{R}$: $s = (x, y, \theta, l)$. The reduction of this line segment to an oriented point is an aspectualization with $\kappa_a(x, y, \theta, l) = (x, y, \theta)$.*

Aspects may span over several features s_i . However, to be able to single out an aspect from a feature vector by aspectualization, it must be guaranteed that no feature refers to more than one aspect. We call this property *aspectualizability*:

Definition 3. *If one or more components of a feature vector exclusively represent one aspect, then we call \mathcal{S} aspectualizable regarding this aspect.*

It is possible to find an aspectualizable and a non-aspectualizable representation that have the same semantics, as the following example shows:

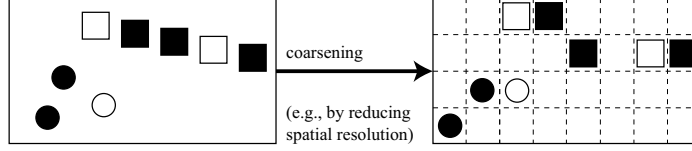


Figure 3: Coarsening reduces the resolution of feature values. Here, coarsening abstracts from coordinates of spatial entities by arranging them into a grid.

Example 2. *The oriented line segment representation in Example 1 can be bijectively mapped from point, angle, and length to two points (x_1, y_1, x_2, y_2) . Then aspectualization cannot single out the length of the line segment, because length is not represented explicitly. It is implicitly given and needs to be calculated by $((x_2, y_2) - (x_1, y_2))$. Thus, \mathcal{S} is not aspectualizable regarding length.*

2.2.2 Coarsening

Aspectualization reduces the number of features, but retains the set of possible values the remaining features can take. When the set of values a feature dimension can take is reduced, we speak of a coarsening (see Figure 3):

Definition 4. *Coarsening is the process or result of reducing the details of information of an observation by lowering the granularity of the input space. Formally, it is defined as a function $\kappa : \mathcal{D}^n \rightarrow \mathcal{D}^n (n \in \mathbb{N})$,*

$$\kappa(s) = (\kappa_1(s_1), \kappa_2(s_2), \dots, \kappa_n(s_n)) \quad (2)$$

with $\kappa_i : \mathcal{D} \rightarrow \mathcal{D}$ and at least one κ_i being not injective.

Example 3. *An important representation in the area of robot navigation is the occupancy grid (Moravec & Elfes, 1985), a partition of space into a set of discrete grid cells. A function $\kappa : \mathbb{R}^2 \rightarrow \mathbb{R}^2, \kappa(x, y) = (\lfloor x \rfloor, \lfloor y \rfloor)$ is a coarsening that maps any coordinate to a grid cell of an occupancy grid.*

2.2.3 Conceptual Classification

Conceptual classification is the most general of the three proposed abstraction facets. It can utilize all components of the input to build new entities (see Figure 4 for illustration):

Definition 5. *Conceptual classification abstracts information by grouping semantically related features to form new abstract entities. Formally, it is defined as a non-injective function $\kappa : \mathcal{D}^n \rightarrow \mathcal{D}^m (m, n \in \mathbb{N})$,*

$$\kappa(s_1, s_2, \dots, s_n) = (\kappa_1(s_{1,1}, s_{1,2}, \dots, s_{1,h_1}), \kappa_2(s_{2,1}, s_{2,2}, \dots, s_{2,h_2}), \dots, \kappa_m(s_{m,1}, s_{m,2}, \dots, s_{m,h_m})) \quad (3)$$

with $\kappa_i : \mathcal{D}^{h_i} \rightarrow \mathcal{D}$ and $h_i \in \{1, \dots, n\}$, whereby $i \in \{1, \dots, m\}$.

Conceptual classification subsumes aspectualization and coarsening, as the following theorems show:

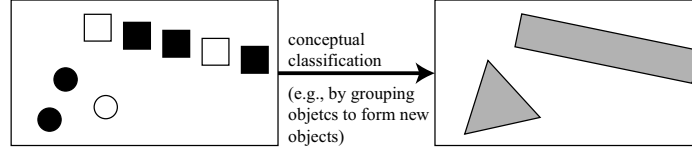


Figure 4: Conceptual classification groups features to form completely new abstract entities.

Corollary 1. *Let κ be a conceptual classification as defined in Definition 5. Then, if all κ_i have the form $\kappa_i : \mathcal{D} \rightarrow \mathcal{D}$ and $m = n$, κ is a coarsening.*

Corollary 2. *Let κ be a conceptual classification as defined in Definition 5. Then, If all κ_i have the form $\kappa_i(s_j) = s_j$, $i \leq j$, $m < n$, and $\kappa_i = \kappa_j \Rightarrow i = j$, then κ is an aspectualization.*

Both theorems are left without a proof, because the theorems follow directly from the definitions.

Example 4. *Data gathered from a laser range finder comes as a vector of distance values and angles to obstacles in the local surrounding, which can be represented as 2-D points in a relative coordinate frame around the sensor. Abstraction of these points to line segments by the means of a line detection algorithm is a conceptual classification.*

2.3 Related Work on Facets of Abstraction

The insight that abstraction can be divided into different categories is widely acknowledged in the literature and has been issued in works from various scientific fields. For example, Stell and Worboys (1999) present a distinction of what they call “selection” and “amalgamation” and formalize these concepts for graph structures. Our definitions of aspectualization and coarsening correspond to selection and amalgamation.

Bertel et al. (2004) also differentiate between different facets of abstraction (“aspectualization versus specificity”, “aspectualization versus concreteness”, and “aspectualization versus integration”), but without giving an exact definition. “Aspectualization versus specificity” resembles our definition of aspectualization, and “aspectualization versus concreteness” to coarsening. However, our definition of aspectualization (Definition 2) is tighter than the one given by Bertel et al.—according to them, aspectualization is “the reduction of problem complexity through the reduction of the number of feature dimensions”. In the framework presented in this paper, the result of aspectualization is also a reduction of feature dimensions, but it is also explicitly required that all the values of the remaining feature dimensions are unchanged.

The notion of *schematization*, which Talmy (1983) describes as “a process that involves the systematic selection of certain aspects of a referent scene to represent the whole disregarding the remaining aspects”, is tightly connected to our definition of aspectualization. If we assume the referent scene to be aspectualizable according to Definition 3, then the process mentioned by Talmy is aspectualization as defined in this chapter.

Herskovits defines the term schematization in the context of linguistics as consisting of three different processes, namely abstraction, idealization, and selection (Herskovits, 1998). Abstraction and selection correspond to our definition of aspectualization, while Herskovits’ definition of idealization is a coarsening according to Definition 4.

2.4 Abstraction and Representation

2.4.1 Expressiveness of Abstraction Paradigms

Aspectualization and coarsening describe two very different processes: The first one reduces the number of features of the input, the latter reduces the variety of instances for every single feature. While aspectualization necessarily reduces the dimensionality of a representation, coarsening preserves dimensionality. Abstraction processes inherently depend on the underlying knowledge representation. However, we will show in the following that both facets of abstraction are equally expressive if we allow for bijective modifications of the underlying representation.

Depending on the representation of the feature vector, coarsening can produce a result that is equivalent to an aspectualization: Let one or more mappings κ_{c_i} in a coarsening be defined as mappings to a single constant value: $\kappa_i = z, z \in \mathcal{D}$. Assume all other mappings κ_{c_i} to be the identity function. Now, consider an aspectualization that retains exactly the components not mapped to single constant values z_i by the coarsening. The target space \mathcal{T} created by this aspectualization results in a linear subspace of \mathcal{S} that is a projection from the resulting space of the coarsening. In other words, the aspectualization has a canonical embedding in the result of the coarsening; and we can find a bijection that maps one into the other. Existence of this process is captured by the following theorem:

Theorem 3. *If κ_a is an aspectualization, then there exists a bijection φ_1 and a coarsening κ_c such that the following diagram commutes:*

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{\kappa_c} & \mathcal{T}' \\ & \searrow \kappa_a & \swarrow \varphi_1 \\ & & \mathcal{T} \end{array}$$

Proof. Following Definition 2, aspectualization conserves those features of (s_1, \dots, s_n) listed in a vector $I^\kappa = (i_1, \dots, i_k), I^\kappa \in \mathbb{N}^k$. Then, the vector $I'^\kappa \in \mathbb{N}^n$ denotes the original position of the remaining features in the source feature vector s :

$$I'^\kappa = (i'_1, \dots, i'_n), \quad i'_j = \begin{cases} j | I_j^\kappa = i & \text{if } \exists j : I_j^\kappa = i \\ 0 & \text{else} \end{cases} \quad (4)$$

Now we chose $\kappa_c : \mathcal{S} \rightarrow \mathcal{T}, \kappa_c(s_1, \dots, s_n) = \kappa_{c_1}(s_1), \dots, \kappa_{c_n}(s_n)$ with

$$\kappa_{c_i}(s_i) = \begin{cases} s_i & \text{if } i \in I^\kappa \\ z & \text{else} \end{cases}$$

and a constant $z \in \mathcal{D}$. Furthermore, we chose $\varphi_1 : \mathcal{T}' \rightarrow \mathcal{T}, \varphi_1(t'_1, \dots, t'_n) = (t'_{i_1}, \dots, t'_{i_k})$ with $i_j \in I^\kappa$ and $\varphi_1^{-1} : \mathcal{T} \rightarrow \mathcal{T}', \varphi_1^{-1} = (\varphi_{1_1}^{-1}, \dots, \varphi_{1_n}^{-1}),$

$$\varphi_{1_i}^{-1}(t_1, \dots, t_k) = \begin{cases} t_{I_i^\kappa} & \text{if } I_i^\kappa \neq 0 \\ z & \text{else} \end{cases}$$

□

The following example illustrates Theorem 3:

Example 5. As in Example 1, an oriented line segment in the plane is represented as a point $(x, y) \in \mathbb{R}^2$, a direction $\theta \in [0, 2\pi]$, and a length $l \in \mathbb{R}$: $s = (x, y, \theta, l)$. The function $\kappa_c(x, y, \theta, l) = (x, y, 1, l)$ is a coarsening, and κ_c delivers essentially the same result as an aspectualization $\kappa_a(x, y, \theta, l) = (x, y, l)$. There exists a trivial bijection between both results.

The other way round, it is also possible to define a bijection that allows for expressing a coarsening by an aspectualization: If \mathcal{S} is a mathematical structure as, for example, a group, then this bijection even maintains the structure of \mathcal{S} , that is, it is an isomorphism:

Theorem 4. If κ_c is a coarsening on a group, for example $(\mathcal{S}, +)$, then there exists an isomorphism φ_2 and an aspectualization κ_a such that the following diagram commutes:

$$\begin{array}{ccc} \mathcal{S} & \xrightarrow{\varphi_2} & \mathcal{S}' \\ & \searrow \kappa_c & \swarrow \kappa_a \\ & \mathcal{T} & \end{array}$$

Proof. Choose $\varphi_2(s) = (s + \kappa_c(s), \kappa_c(s))$, $\varphi_2^{-1}(t_1, t_2) = t_1 + (-t_2)$ and $\kappa_a(t_1, t_2) = t_2$, and define (\mathcal{S}', \oplus) with $\mathcal{S}' = \text{Image}(\varphi_2)$ and $t \oplus u = \varphi_2(\varphi_2^{-1}(t) + \varphi_2^{-1}(u))$ for each $t, u \in \mathcal{S}'$. Checking that (\mathcal{S}', \oplus) is a group and φ_2 a homomorphism is straightforward. \square

Example 6. Coordinates $(x, y) \in \mathbb{R}^2$ can be bijectively mapped to a representation $(\lfloor x \rfloor, x - \lfloor x \rfloor, \lfloor y \rfloor, y - \lfloor y \rfloor)$ which features decimal places separately. The function $\kappa(x, x', y, y') = (x, y)$ is an aspectualization with the same result as the coarsening in Example 3.

Theorems 3 and 4 show that we can both achieve the result of an aspectualization by bijective translation of a coarsening as well as we have a bijection to transform any representation such that we can achieve the result of a coarsening by an aspectualization.

Stell and Worboys (1999), among others, describe aspectualization and coarsening (or selection and amalgamation, as they call it), as being “conceptually distinct”. However, regarding Theorems 3 and 4, this cannot be agreed on. The distinctness only applies to the *process* of abstraction and not the *result*. Different abstraction paradigms, even if describing distinct processes, can have the same effect and lead to an identical outcome. Whether a specific abstraction paradigm is applicable depends on the given representation in the source space. This means that if we want to apply a specific abstraction paradigm, we must invest in finding a representation that allows for that. In other words, the choice of an abstraction paradigm is tightly coupled with the choice of the state space representation: It is always possible to find a bijection such that we can use the abstraction paradigm that is most convenient for the task at hand.

2.4.2 Accessibility of a Representation

Different kinds of representations allow for expressing the same knowledge, but they differ in the *accessibility* of the relevant information. When choosing a representation for a certain purpose, we argue for using aspectualizable representations and encode relevant details as aspects, that is, bringing them to the feature level. Having important information as separate features in a feature vector

allows them to be easily accessed both by humans and algorithms: aspectualization is a computationally and cognitively simple selection process, and humans are able to access relevant information easily.

Aspectualizable feature vectors foster the processing of a representation with regard to the relevant features encoded. For example, identity of aspects can easily be detected—our concept of structure space policies (Section 3.3.3) and the APSST algorithm (Section 4.2.1) make use of this property. Algorithms often operate on the feature level, and if we want them to exploit relevant information, then the representation must be aspectualizable. One example of an approach working on singular features is the irrelevant state variable algorithm introduced by Jong and Stone (2005). Also, Frommberger (2008) achieves generalization in a learning task by storing a Q-function operating on an aspectualizable representation in a CMAC.

Hence, it is beneficial to create aspectualizable knowledge representations from perceived data. One possibility is to transform coarsenings into aspectualizations (Theorem 4). Another possibility is to use abstraction itself; in particular conceptual classification is a powerful means to create new entities in an aspectualizable manner. Often, aspectualizable representations can be constructed by applying abstraction techniques.

3 Abstraction in Agent Control: Exploiting Structure for Knowledge Transfer

In this section, we show how abstraction can be utilized in the context of *agent control*, in particular, how abstraction enables knowledge reuse in reinforcement learning tasks. An intelligent agent requires a knowledge representation that provides support in two essential regards: First, to exploit similarities within the environment and second, to be able to process elementary details of its input in a reusable manner. The next sections describe these capabilities and how aspectualizable state space representations can be utilized for them.

In the following, we first introduce *Markov Decision Processes* (MDPs) as a means to model dynamic environments with an interacting agent. In this work, we will regard problems defined as an MDP.

3.1 Markov Decision Processes and Reinforcement Learning

An MDP $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$ consists of a set of system states \mathcal{S} , a set of actions \mathcal{A} , a transition function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ denoting a probability distribution that performing an action a at state s will result in state s' and a reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that assigns a reward (or reinforcement) for taking an action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$. Subsequent states in an MDP are fully described by the probability distribution T , so they depend only on the current state and the executed action. In particular, the next state does not depend on the way the current state was reached—past states or actions are irrelevant. This property is called the *Markov property*. The solution to an MDP is an optimal policy π^* that maximizes the rewards achieved over time.

Reinforcement learning (Sutton & Barto, 1998) is a popular and effective method to solve an MDP. In particular, we look at the reinforcement learning algorithm *Q-learning* (Watkins, 1989) in this work. At each point in time, the agent is in a particular state $s \in \mathcal{S}$, and the agent's view on this state is represented by a feature vector. Based on this information the agent draws decisions which action a from a set of possible actions \mathcal{A} to take in order to reach its goal. The outcome of

Q-learning is a Q-function $Q(s, a)$ that assigns to any state-action pair (s, a) the overall expected reward over time when starting in s and executing action a . From that Q-function, a *policy* π can be derived by always executing the action with the highest Q-value: $\pi(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$.

Under friendly circumstances, Q-learning converges to an optimal Q-function $Q^*(s, a)$ that returns the highest possible expected reward for any state-action pair (s, a) and thus established an *optimal policy* π^* .

In this work, we focus on goal-directed tasks. In an MDP setting this means, that there is a set $\mathcal{S}^+ \subset \mathcal{S}$ that $R(s, a)$ delivers a positive reward when taking action a in state s results in the agent being in a state in \mathcal{S}^+ . Any $s \in \mathcal{S}^+$ is a *goal state*.

3.2 Structural Similarity and Knowledge Transfer

A key capability of an autonomous agent is to generalize over similarities in the environment it is navigating in. Different situations or places within a task are often not distinct in every respect. Rather, they share some commonalities: Knowledge gained in one situation may be useful in another, and action selection to solve the task may partially lead to related actions of the agent. Put differently: Action selection depends on properties of the state space that can be encountered in various different situations. This kind of similarity of states can also hold across different tasks.

We call recurring properties that are significant for action selection the *structure* of a domain. For example, the structure of indoor office environments is given by the spatial arrangement of obstacles like walls. We call state spaces that share structures within their state spaces *structurally similar*. Problems or tasks are structurally similar if their state spaces are structurally similar and policies solving the task exploit structure in the same way. For example, if in one task the goal of the agent is to follow a corridor and in the other task to bump into walls, then the tasks are *not* structurally similar. But if the new goal involves following the corridor in the opposite direction, then structural similarity is given, as the agent has to cope with the structure in the same way: staying in the middle of the walls.

When action selection is based on structural properties it remains the same across different situations or tasks with the same structure. In agent control, we always encounter situations where the agent has incomplete knowledge about which action to take to reach a certain goal. In this context knowledge reuse is an essential capability: It empowers the agent to make use of similarities by utilizing previously gained knowledge for mastering the current challenge. When the agent is able to notice that a state it is in is structurally similar to a state of which it already gathered knowledge, then it can base a new decision on this structural knowledge. The agent can abstract from the concrete state and rely on the structure of a situation: In this context, structure is a result of an abstraction of the current state representation.

3.3 Aspectualizable State Spaces

3.3.1 A Distinction of Different Aspects of Problems

In structurally similar environments, we experience a locally identical behavior of the agent. For example, the actions that a robot employs to traverse a corridor are the same, regardless where the corridor is positioned (it can be in the same environment or in a different one). So the agent's behavior depends on two separate aspects—one that is tightly coupled to the goal to be reached and one that is influenced by structural elements of the state spaces. We distinguish two aspects within the behavior of the agent:

Goal-directed behavior towards a goal state highly depends on the specific task to be solved. In general, resulting actions at distinct states are different for different goal states or in different environments. Goal-directed behavior is specific for the task at hand, the corresponding knowledge about the behavior is *task-specific knowledge*.

Generally sensible behavior regards the structure of the environment, so it is more or less the same in structurally similar state spaces. It does not depend on a goal to reach, but on structural characteristics in the environment that demand a certain behavior. Generally sensible behavior is independent of the task at hand, the corresponding knowledge about the behavior is *task-independent knowledge*.

Goal-directed and generally sensible behavior are not completely independent of one another: Reaching a target location requires some sort of generally sensible behavior. Knowledge of generally sensible navigation behavior is a good foundation for developing goal-oriented strategies. There must be a generally sensible behavior in every successful goal-oriented strategy, otherwise a goal state would not be reached—positive rewards can only be reached if this kind of behavior is followed. Therefore, generally sensible behavior is an excellent candidate for knowledge transfer, because it is existent and useful in any structurally similar task. In the following, we show how this behavior can be singled out from an agent’s policy by the means of an appropriate representation of the state space.

3.3.2 Structure Space and Task Space Aspectualization

Generally sensible behavior critically depends on structural characteristics of the environment. To enable computational operations needed for knowledge transfer on these characteristics, we want them to be easily accessible within the state space representation. As we have argued in Section 2.4.2, it is especially aspectualization that fosters accessibility.

Thus, within the process of state space design—which is usually driven by the designer’s domain knowledge about the task to solve—we argue for explicitly modeling the structure of an environment as aspects, that is, a set of features within the state space representation. We call the feature space generated by these features *structure space* (\mathcal{S}_S). If \mathcal{S} is aspectualizable with respect to those features, that is, if they are a subset of the overall state space representation $s = (s_1, s_2, \dots, s_n)$, we call this a *structure space aspectualizable* state space, and there exists a function $\text{ssd} : \mathcal{S} \rightarrow \mathcal{S}_S$ that returns the features of structure space. Generally sensible behavior operates on structure space.

We call the space generated by the remaining features *task space* (\mathcal{S}_T); let those features be delivered by a function $\text{tsd} : \mathcal{S} \rightarrow \mathcal{S}_T$. Task space relates to goal-directed behavior. Summed up, we assure that the state space \mathcal{S} is represented as a Cartesian product of task and structure space representation $\mathcal{S}_T \times \mathcal{S}_S$ such that we have a state representation $s = (\text{tsd}(s), \text{ssd}(s))$.

The distinction between task space and structure space relates to the framework of problem space and agent space (Konidaris, 2006) that had been presented concurrently with our earlier work on this issue (Frommberger, 2006). In the problem/agent space framework, problem space stands for the set of features defining the state space for the overall task. An agent space consists of a (usually disjoint) set of attributes that are shared under the same semantics by different tasks. The most important distinction between both frameworks is that structure space also refers to similarities at different places *in the same* environment. Thus, it does not only relate to transfer learning, but also enables for generalization by exploiting structural commonalities *within* one learning task (Frommberger,

2008). In the work presented here, a structure space aspectualizable state space is a special case where an agent space representation becomes part of the problem space representation.

3.3.3 Structure Space Policies

If we now want to single out a generally sensible behavior of the agent, we aim at finding a policy that *exclusively* depends on structural properties of the environment, that is, it only operates on structure space \mathcal{S}_S . We call such a policy a *structure space policy* π_S . In contrast to regarding the complete state s , it draws the decision which action to take based on structural features only and thus implements a generally sensible behavior as described in Section 3.3.1.

A structure space policy, once derived, is applicable in any structurally similar task and enables the agent to operate in a reasonable way. From a state s , the structure space description $\text{ssd}(s)$ can be achieved by a simple aspectualization, and $\pi_S(\text{ssd}(s))$ immediately delivers an action according to a generally sensible behavior. Thus, a structure space policies can be transferred to structurally similar tasks: In structurally similar structure space aspectualizable state spaces a structure space policy is immediately applicable—no transfer algorithm is needed, as the structure space description is identical over the tasks and can be extracted from a feature vector without any effort.

3.3.4 Identifying Structure Space

One still open question is how to identify a structure space for a given task. If we assume that a subset of the features of a state (s_1, \dots, s_n) encodes the structure of a domain, then the problem is reduced to the question which features s_i are irrelevant for a structure space policy, so we could examine a task that only operates on structure space and draw conclusions from that. Jong and Stone (2005), for example, provide statistical methods to discover irrelevant features within a state representation. Another possibility would be to empirically test generated structure space policies (as described in Section 4.2) for certain variations of state variables while (or after) solving the original problem.

The crucial problem there is the assumption that structure is encoded in separate state variables, that is, the state space is (at least partially) structure space aspectualizable. In general, this does not need to be the case (see Example 4.2). Thus, at this point, we still cannot withdraw the explicit representation of structural features from the the responsibility of the system designer.

4 Generating Structure Space Policies in Reinforcement Learning

In this section we demonstrate the use of aspectualizable state space representation for knowledge transfer. We show how to generate a structure space policy in structure space aspectualizable state spaces from a policy that has been acquired with reinforcement learning.

4.1 Reinforcement Learning and Knowledge Transfer

In general, a strategy π learned with Q-learning is bound to the goal state the agent learned to reach. For mastering another task with, for example, different goal states, the whole strategy would need to be re-learned from scratch—the knowledge gained in the first learning task and represented in the Q-function is not applicable under changed conditions.

In the area of machine learning, the challenge of re-using parts of the gained knowledge of a learning task and transferring it to another one is labeled *transfer learning*. Transfer learning can be divided into two differently challenging categories: approaches that solve new tasks within a given domain and approaches that also work in a different domain. We call the first *intra-domain transfer*, while the latter is termed *cross-domain transfer* (Taylor & Stone, 2007). In an MDP we speak of the same domain if state space, action space, and transfer probabilities are identical, but the task to solve is a different one. That is, intra-domain transfer tackles problems that differ only in the reward function.

Examples of intra-domain transfer methods are the approaches that use temporal abstraction, as for example skills (Thrun & Schwartz, 1995), options (Sutton et al., 1999), or MAXQ (Dietterich, 2000). Also, policy reuse (Fernández & Veloso, 2006) has to be named in this context. Examples of cross-domain transfer learning are the AI^2 algorithm (Torrey, Shavlik, Walker, & Maclin, 2006), rule transfer (Taylor & Stone, 2007), or the approaches based on the problem/agent space framework (Konidaris & Barto, 2006, 2007) that has already been mentioned in Section 3.3.2.

Especially cross-domain transfer is interesting as it involves applying gained knowledge in situations and in environments that are unknown to the artificial agent and enable it to show an anticipatory and appropriate behavior under conditions that have not been encountered before. Apart from the problem/agent space framework, all the named cross-domain transfer methods have in common to require an explicit a priori definition of relationships between source and target task. In contrast to that, the work presented in this paper encodes this relationship implicitly in the state representation by choosing aspectualizable structural features that maintain their semantics over different tasks. That means that only the structure of the target task has to be known a priori, not its details. Our approach differs from the problem/agent space approach in making the commonalities between tasks an aspectualizable part of the state representation which allows for straight-forward utilization in a target task.

4.2 A Posteriori Structure Space Aspectualization

Extracting a structure space policy (Section 3.3.3) from a learned policy is a kind of cross-domain transfer learning: A structure space policy π_S achieved from a learned policy in a *source task* can be immediately applied in a structurally similar *target task* and implements a generally sensible behavior there.

4.2.1 The APSST Algorithm

In the following, we present an a posteriori method that operates on a successfully learned Q-function Q and policy π . It uses to generate a structure space policy π_S with a function $Q_S : \mathcal{S}_S \rightarrow \mathbb{R}$ operating on the structure space of the source task. We refer to this procedure as *a posteriori structure space transfer* (APSST). The prerequisite for this algorithm is the existence of a structure space aspectualizable state space $\mathcal{S} = \mathcal{S}_T \times \mathcal{S}_S$. Furthermore, \mathcal{S} has to be discrete, a property that is beneficial for any reinforcement learning task and can be achieved by a proper use of abstraction (cf. Section 5.2.1).

To generate a structure space policy, information on states with the same structure space descriptor is summarized. This is achieved by *Q-value averaging*: To get the new Q-function $Q_S(\bar{s}, a)$ ($\bar{s} \in \mathcal{S}_S$, $a \in \mathcal{A}$), we calculate the arithmetic means over all the Q-values of all state-action pairs $(s, a) =$

$((\text{tsd}(s), \bar{s}), a)$ ($s \in \mathcal{S}$) with the same structure space representation $\bar{s} = \text{ssd}(s)$:

$$Q_S(\bar{s}, a) = \frac{1}{|\{x \in \mathcal{S}_T | Q((x, \bar{s}), a) \neq 0\}|} \sum_{y \in \mathcal{S}_T} Q((y, \bar{s}), a) \quad (5)$$

We only regard states where we have information on, that is, the Q-value does not equal 0.

In reinforcement learning the system is usually trained with discounted rewards (the reward estimation is diminished over time) or with negative reinforcement for any action taken. Both options result in Q-values that are high for states “near” the goal state and decrease with distance from it. As Equation (5) averages Q-values, states in the vicinity of the goal have a bigger impact on the generated strategy. To account for this, we extend Equation (5) by a weighting factor to even out the differences in Q-values for states in different distances to the goal state. We can compare the Q-values for all the actions a at state s and normalize them with regard to the absolute value of the maximum Q-value $|\max_{b \in \mathcal{A}} Q(s, b)|$. Then for the normalized values holds

$$-1 \leq \frac{Q(s, a)}{|\max_{b \in \mathcal{A}} Q(s, b)|} \leq 1 \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \quad (6)$$

Thus, the overall equation for a posteriori structure space aspectualization is

$$Q_S(\bar{s}, a) = \frac{1}{|\{x \in \mathcal{S}_T | Q((x, \bar{s}), a) \neq 0\}|} \sum_{y \in \mathcal{S}_T} \frac{Q((y, \bar{s}), a)}{\max_{b \in \mathcal{A}} |Q((y, \bar{s}), b)|} \quad (7)$$

for every $\bar{s} \in \mathcal{S}_S$.

The presented algorithm sums up over *all* task space observations. However, usually only a small fraction of the theoretical task space is physically existent. Furthermore, it may not have been completely explored. So in practice it is only necessary to keep track of the visited state-action pairs and to regard those. An algorithmic notation of APSST is given in Algorithm 1. Its runtime is linear to the number of states visited.

4.2.2 Using APSST Generated Policies for Transfer Learning

The generated structure space Q-function Q_S delivers the average over the (normalized) Q-values of the original Q-function Q for each (\bar{s}, a) . From this it is possible to infer which action a is most promising with respect to the average of all states s sharing the same structure $\text{ssd}(s) = \bar{s}$. If, according to Q , an action a_0 is the best choice in every state with a structure \bar{s} (that is, $Q(s, a_0) \geq Q(s, a) \forall a \in \mathcal{A}$) then a_0 is also the best action for a structure \bar{s} according to Q_S , that is, $Q_S(\bar{s}, a_0) \geq Q_S(\bar{s}, a) \forall a \in \mathcal{A}$.

As Q_S is generated by averaging different Q-values all over the state space, Q_S is decoupled from the semantics of concrete reward expectations. However, $\arg\max_a Q_S(\bar{s}, a)$ returns the action that showed the highest average reward expectation for a given structure. This selection is independent from whether the original Q-values have been high or low, as only the Q-values of different actions at a certain structure are compared. Thus, greedily following those actions implements a generally sensible behavior with respect to structure space: The agent executes the action that—on the average—proved to be most successful in the source task for all states sharing the given structure.

Knowledge contained in a structure space policy Q_S can be used to improve the learning performance in a new, but structurally similar task. For example, it could be used to generate shaping

Algorithm 1 A Posteriori Structure Space Aspectualization (APSST)

Require: \mathcal{S}_v contains all visited states, T is an empty associative table

```
 $M_S \leftarrow \emptyset$   
for all  $s \in \mathcal{S}_v$  do  
   $i \leftarrow 0$   
  for all  $a \in \mathcal{A}$  do ▷ look at the Q-values for each action  
     $v[i] \leftarrow Q(s, a)$   
     $i \leftarrow i + 1$   
  end for  
   $m \leftarrow \max_j |v[j]|$  ▷ maximum for weighting factor  
  for all  $a \in \mathcal{A}$  do  
     $Q_S(\text{ssd}(s), a) \leftarrow Q_S(\text{ssd}(s), a) + \frac{1}{m} Q(s, a)$   
  end for  
   $M_S \leftarrow M_S \cup \text{ssd}(s)$  ▷ store processed structure  
   $T[\text{ssd}(s)] \leftarrow T[\text{ssd}(s)] + 1$  ▷ count appearance  
end for  
for all  $\bar{s} \in M_S$  do ▷ average Q-values  
   $Q_S(\bar{s}) \leftarrow Q_S(\bar{s})/T[\bar{s}]$   
end for
```

rewards (Dorigo & Colombetti, 1994; Konidaris & Barto, 2006). Another possibility, which we exploit in this work, follows the idea of Sutton and Barto (1998, p. 56) that existing prior knowledge (the structure space policy) can most conveniently be imparted in an initial Q-function. This approach has already been successfully exploited for transfer learning tasks (Taylor, Stone, & Liu, 2005). In this work, we use Q_S gained in a first task to initialize the Q-function Q' in the second task. Whenever the agent observes a state s that it has never visited before, the corresponding Q-value $Q'(s, a)$ is initialized with $\frac{1}{K} Q_S(\text{ssd}(s), a)$ for any $a \in \mathcal{A}$. This results in a form of guided exploration: The agent executes a generally sensible behavior at this state instead of choosing a random action. Scaling down the Q-values by a constant factor $\frac{1}{K}$ ($K \gg 1$) shall ensure that initial values are lower than the optimal Q-values to be learned. This is necessary since in the new task Q_S is decoupled from the semantics of concrete reward expectations and overly high reward expectations introduced from Q_S will disturb the learning progress. An good value for K can be computed from maximum reward, discount factor, and maximum number of steps in each episode.

In cases where $\max Q(\bar{s}, a)$ does not vary much for different $a \in \mathcal{A}$, action selection in the learned policy is ambiguous with respect to structure \bar{s} . Thus, action selection mainly depends on task space information here. However, structures with this property do not inflict the transfer learning method described above: The initial Q-values of the new Q-function still hint at a sensible behavior, and even if $Q_S(\bar{s}, a)$ is the same for all $a \in \mathcal{A}$.

5 Structure Space Policy Transfer from Simulation to a Real Robot

In this section we show in a detailed case study how abstraction becomes an essential means to knowledge transfer. We regard a very difficult form of knowledge transfer: the transfer of a policy learned in the simulator to a real robotics platform. This example demonstrates that on the one

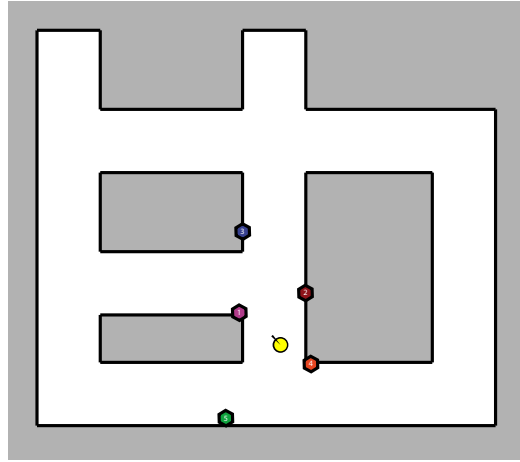


Figure 5: A robot in a simulated office environment with some landmarks around it

hand abstraction allows us to cope with real world concepts in the same way as with simulated ones and on the other hand that the transfer of knowledge benefits from aspectualizability of an abstract state space representation. In particular, we show that the use of an aspectualizable representation empowers the ability to adapt to unknown, yet structurally similar environments.

5.1 The Task

In the learning task we consider in this case study an autonomous robot is requested to find a certain location in a simplified and idealized simulated office environment (see Figure 5). The environment consists of walls, represented by line segments, and uniquely identifiable landmarks the robot can perceive when they are in its field of vision. At the start of the learning task the world is completely unknown to the agent—no map is given and no other information is provided.

A number of primitive actions (a subset of going straight, back, and various turns) can be taken by the robot, but it has no idea of the implications of its actions, that is, no model of the control task is given. No built-in collision avoidance or any other navigational intelligence is provided. The robot is assumed to be able to perceive walls around it within a certain maximum range.

The agent’s task is to “find” a certain location s^* within the environment and drive towards it. A positive reward will be given when a goal state $s^* \in \mathcal{S}^*$ is reached and a negative one if the agent collides with a wall.

5.2 Learning a Policy in Simulation

First, a successful policy has to be learned in the simulator. The presented learning task is a complex one, because it is large and continuous, and RL is known to suffer from performance problems under these conditions. Often, this is tackled by value function approximation (Kaelbling, Littmann, & Moore, 1996), but due to the inhomogeneity of the state space at the position of walls, this approximation is crucial, and spatial environments can be arbitrarily large. Thrun and Schwartz (1995) argue that for such complex problems it is necessary to discover the structure of the world and to abstract from its details. Thus, to learn a policy for this task, we use an abstract state space representation that is generated by applying abstraction methods to create a discrete structure space

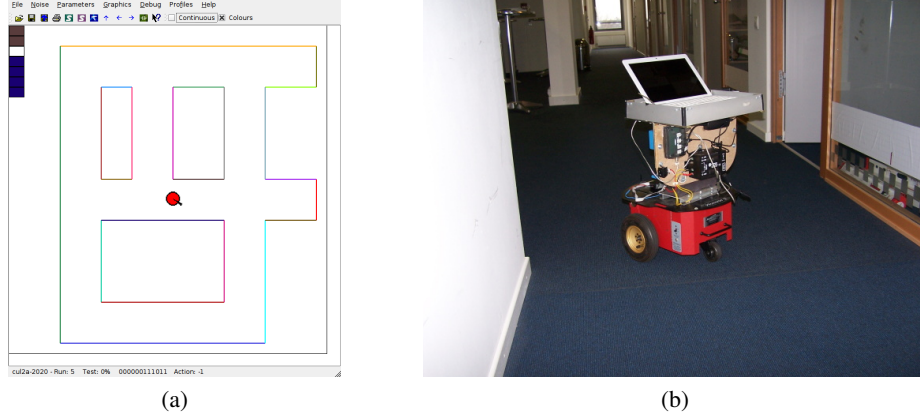


Figure 6: A screenshot of the robot navigation scenario in the simulator where the strategy is learned (a) and a Pioneer-2 robot in an office building, where the strategy shall be applied (b). The real office environment offers structural elements not present in the simulator: open space, uneven walls, tables, and other obstacles.

aspectualizable state space.

It must be noted that under state abstraction solving an MDP with RL in general does not result in optimal policies anymore. Even if a chosen abstraction preserves the optimal policy π^* , Q-learning can result in suboptimal policies (McCallum, 1995), and even convergence cannot be ensured (Li, Walsh, & Littman, 2006). The underlying system is no MDP anymore: ψ collapses several different states to one single observation. For non-redundant state representations, this violates the Markov property, thus the system becomes a *Partially Observable Markov Decision Process* (POMDP) (Lovejoy, 1991). In general, solving POMDPs is a hard and—for large state spaces—open issue.

The approach we use in this work to find a policy for the given task is simply to ignore the partial observability property and solve the resulting problem with Q-learning as if it was an MDP. Of course, the quality of a solution using this approach critically depends on how tightly the system dynamics of POMDP and underlying MDP are coupled. The state space representation we use for this scenario, which is introduced in the next section, has been shown to find successful policies for the given task. It outperforms metrical representations that rely on distances or absolute coordinates with regard to learning speed and robustness (Frommberger, 2008).

5.2.1 Landmark-enriched RLPR

In order to obtain a structure space aspectualizable state space, structural entities must be represented separately. Office environments as depicted in Figure 6 are characterized by walls, which can be abstracted as line segments in the plane. Foremost it is relative position information of line segments with respect to the robot’s moving direction that defines structural paths in the world and leads to sensible action sequences for a moving agent. Thus, for encoding structure space, we use the qualitative *Relative Line Position Representation RLPR* (Frommberger, 2008). For RLPR, the space around the agent is partitioned into bounded and unbounded regions R_i (see Figure 7). Two functions $\bar{\tau} : \mathbb{N} \rightarrow \{0, 1\}$ and $\bar{\tau}' : \mathbb{N} \rightarrow \{0, 1\}$ are defined: $\bar{\tau}(i)$ denotes whether there is a line segment detected within a region R_i and $\bar{\tau}'(i)$ denotes whether a line spans from a neighboring re-

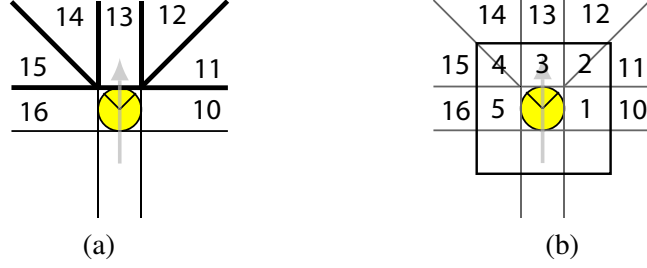


Figure 7: Neighboring regions around the robot in relation to its moving direction. Note that the regions R_1, \dots, R_5 in the immediate surroundings (b) overlap R_{10}, \dots, R_{16} (a). For the robot experiment, only the thick drawn boundaries in (a) are regarded for building the representation.

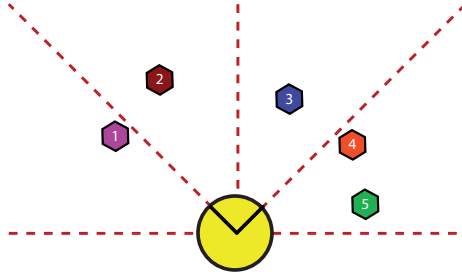


Figure 8: Four sectors and detected landmarks around the robot.

gion R_{i+1} to R_i . $\bar{\tau}_i$ is used for bounded sectors in the immediate vicinity of the agent (R_1 to R_5 in Figure 7(b)). Objects that appear there have to be avoided in any case. The position of detected line segments in R_{10} to R_{16} (Figure 7(a)) is helpful information to be used for general orientation and mid-term planning, so $\bar{\tau}'$ is used for R_{10} to R_{16} . So RLPR encodes the structure space representation ψ_S of an original state $s \in \mathcal{S}$ like this:

$$\psi_S(s) = (\bar{\tau}(R_1), \dots, \bar{\tau}(R_5)), \bar{\tau}'(R_{10}), \dots, \bar{\tau}'(R_{16})) \quad (8)$$

This abstraction from line segments in the simulator to a vector of RLPR values is a conceptual classification.

For representing task space it is sufficient to encode a *qualitative* position of the agent within the world. To do this, representing a position by the circular order of visible landmarks has shown to be an appropriate means (Schlieder, 1993). In our scenario, this can be achieved by regarding detected landmarks in circular sectors around the robot (see Figure 8) or by looking at a sequence of detected walls c_i at a number of discrete angles around the robot.¹ In this work, the latter variant was used with a number of 7 angles. As a result, task space is represented by a sequence of 7 task space features. So, by again employing a conceptual classification, we achieve the task space representation $\psi_T(s)$ of an original state $s \in \mathcal{S}$:

$$\psi_T(s) = (c_1, \dots, c_7) \quad (9)$$

¹The used simulator supports detection of walls by marking each wall with a unique color.

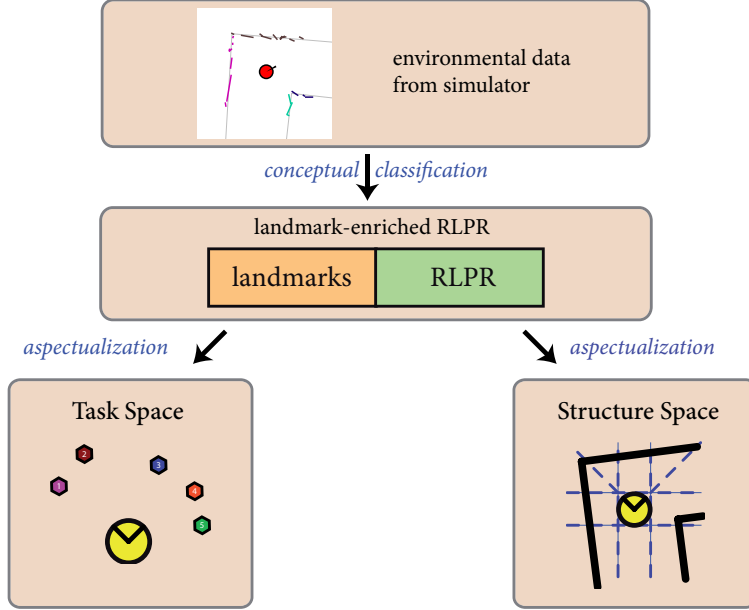


Figure 9: Abstraction principles used to build a structure space aspectualizable representation

We now define an observation for any $s \in \mathcal{S}$ by concatenating task space and structure space representation:

$$\psi(s) = (\psi_l(s), \psi_r(s)) = (c_1, \dots, c_7, \bar{\tau}(R_1), \dots, \bar{\tau}(R_5), \bar{\tau}'(R_{10}), \dots, \bar{\tau}'(R_{16})) \quad (10)$$

We call this spatial representation *landmark-enriched RLPR* (le-RLPR). The emerging observation space is discrete and structure space aspectualizable and thus allows for applying APSST.

5.2.2 Training and Transfer

The system was trained in the environment depicted in Figure 6a, using le-RLPR as described above. The longest corridor in this environment has a length of 15 meters. The simulated robot is a circular object with a diameter of 50 cm. It is capable of performing three different basic actions: moving forward and turning to the left and to the right. The forward movement atomically moves the robot 2.5 cm and the turns rotate the robot by approximately 4° while additionally also moving the robot 2.5 cm forward. A chosen action is repeated until the perceived representation changes (and the discount factor γ is adapted accordingly). The robot is able to perceive walls and landmarks around it within a range of 3.5 meters. Objects or landmarks beyond this range are disregarded. We have chosen Q(λ) learning (Watkins, 1989) with a step size of $\alpha = 0.05$, a discount factor of $\gamma = 0.98$, and $\lambda = 0.9$. For exploration we use a ϵ -greedy exploration strategy with an exploration rate of $\epsilon = 0.15$. The reward function R is designed such that the agent receives a positive reward of 100 for reaching one of the goal states (a certain area within the environment) and a negative one of -100 for colliding with a wall.

From the learned policy in this environment—the source task—we can derive structure space policies with APSST. To evaluate their quality, we applied them to new, structurally similar environments (see Figure 10). Note that the worlds in Figures 10b and 10c contain structural elements (non- 90° turns, for example) that have not been contained in the source task. In these new worlds,

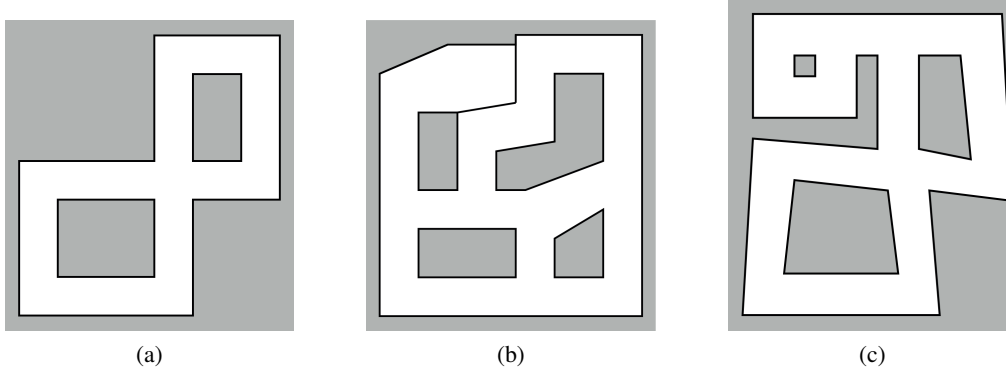


Figure 10: Structurally similar environments used for evaluating the performance of structure space policies.

episode	World (a)	World (b)	World (c)
500	5.7	0.5	0.0
1,000	11.1	8.4	0.2
2,500	59.2	62.2	31.5
5,000	99.2	94.3	58.7
10,000	97.5	77.5	97.2
30,000	99.4	99.7	82.0
50,000	100	99.9	96.2

Table 1: Performance of structure space policies generated with APSST in the environments shown in Figure 10: percentage of collision-free test runs.

no goal area was defined and no learning took place anymore. The agent started from 1,000 starting positions randomly spread over the corridors. A test run in the destination task was regarded as successful if the robot was able to navigate within the corridors without colliding after 1500 action selections.² Test runs in the destination tasks have been performed after a varying number of learning episodes in the source task ranging from 500 to 50,000. Table 1 shows the percentage of successful runs in the environments shown in Figure 10. Depending on the complexity of the target task, the structure space policy shows a very good performance already after 5,000 to 10,000 episodes. Figure 11 shows sample trajectories of the agent in the world in Figure 10b, following a structure space policy generated with APSST after 40,000 episodes of training.

5.3 Learning a Task in a New Environment

In this section we show how a structure space policy generated with APSST can be used for a transfer learning task. An agent learns a policy for a goal-oriented learning task in environment A, and this knowledge improves the learning performance for a new goal-oriented task in environment B.

²The number of time steps had been chosen approximately such that the robot would have been able to explore the environment approximately twice in such a test run. Note that basic actions are repeated if the perception does not change.

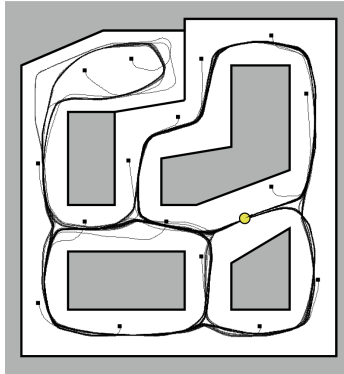


Figure 11: Trajectories of the agent following a structure space policy generated with APSST.

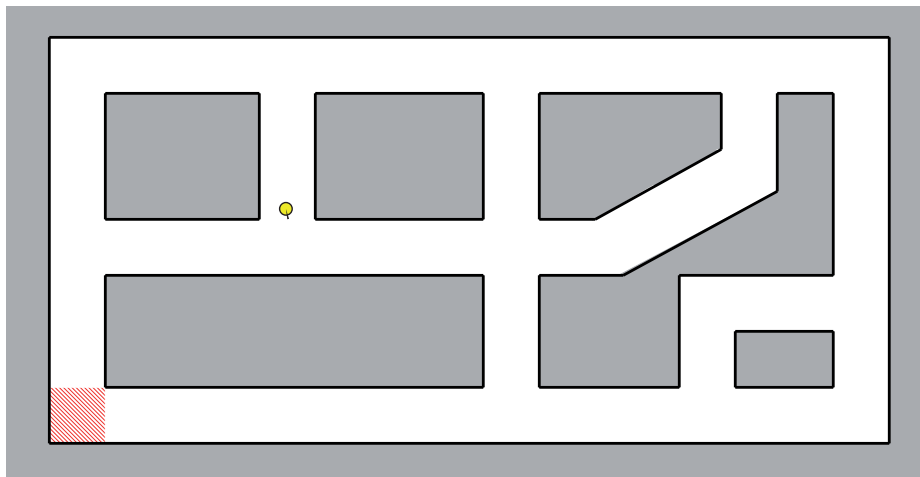


Figure 12: A larger environment where the transfer experiment took place. The longest corridor has a length of 31.5 meters. The goal area is marked.

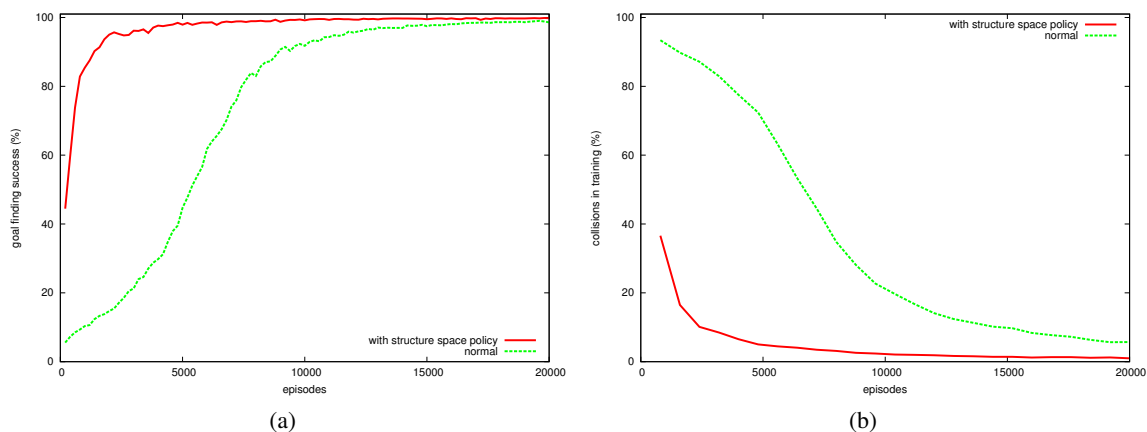


Figure 13: Learning performance with and without incorporation of a structure space policy as prior knowledge: Percentage of episodes reaching a goal state (a) and number of collisions in training (b). The graphs show the average over 30 learning trials

In particular, a policy π was learned for 50,000 episodes in the environment depicted in Figure 6a under the conditions described in Section 5.2.2 with $\alpha = 0.01$ and $\gamma = 0.98$. In each learning episode, the robot started from one of 1,000 pre-defined starting position randomly distributed within the environment. Then, a structure space policy π_S with a Q-function Q_S was derived from π with APSST. A new goal directed learning task was located in the environment depicted in Figure 12, and Q_S was used to initialize the new Q-function in the target task as described in Section 4.2.2 with a scaling value of $K = 10^5$. As a result, the agent follows the learned structure space policy in unknown circumstances, leading to a generally sensible exploration behavior and improved learning performance.

The result of test episodes starting from 1,000 starting positions disjoint from those used in training and executing the greedy policy is depicted in Figure 13: Without using of π_S as prior knowledge, the agent needed more than 15,000 episodes to reach a stable near-100% success, while with the help of the structure space policy this could be achieved in about 7,000 episodes. Furthermore, the agent showed significantly less collisions during training: After 10,000 episodes, the robot did hardly collide anymore with prior knowledge, while it collided in more than 20% of the episodes without.

5.4 From Simulation to Real World Environments

A structure space policy π_S is applicable in any structurally similar environment, because structurally similar situations lead to the same spatial representations even in different environments when relying on π_S . In the following, we will show that this also holds for the case where the environment is not a simplified simulation but a physical spatial environment with a real robot operating in it.

The robot used for the case study in this work is a Pioneer-2 platform (see Figure 6b) equipped with a laser range finder. It is a wheeled robot with two differential motors driving the front wheels. A laptop computer is mounted on the robot to run the control software.

The key issue in knowledge transfer is a shared structure space representation. To achieve that,

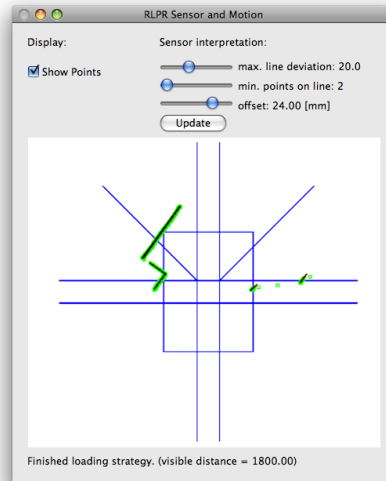


Figure 14: Screenshot: Abstraction to RLPR in the robot controller. Depicted are the RLPR grid, data from the laser range finder (green points), and fitted line segments (red). This output refers to the robot’s position shown in Figure 6b.

both the simulated and the real robot shall operate on RLPR. While this comes easy in the simulator, we must add some efforts on the real system.

RLPR is based on straight line segments, as the simulator provides them. But in real world environments, we never encounter such simple structures. Walls are not perfectly straight, they have niches and skirting boards, and so on. Furthermore, there are obstacles, for example furniture like cupboards, chairs, and tables. Laser range finders detect obstacles around the robot. By measuring laser beams reflected by obstacles one obtains a sequence of (in our case) 361 points in local coordinates. To abstract from those points to line segments we use the well-known iterative split-and-merge algorithm to fit lines to scan data (see, for example, Gutmann, Weigel, and Nebel (2001)). As we have seen in Example 4, this is a conceptual classification. Due to the strong abstraction of RLPR, an overly precise line fitting is not important. With respect to parameters we choose them in a way to make sure that all obstacles detected by the laser range scanners get represented by lines, even if this is a crude approximation.

The detected line configuration is then translated every 0.25 seconds to RLPR such that we receive an RLPR representation of the surroundings of the real robot (see Figure 6 for a screenshot from the robot control software). Now, the robot operates on the same spatial representation as the structure space policy derived from a learned policy in a simulator. Figure 15 summarizes the flow of abstraction in simulator and real environment.

With regard to the agent’s actions, we must take into account that the step-wise motion actions used in the simulator cannot easily be copied to the robot. Thus, movement is smoothed by averaging the k most recent wheel speed commands to avoid strong acceleration/deceleration which the differential robot drive cannot handle well. We averaged over the last 8 actions, so with a 0.25 second interval for sending wheel commands this yields a time of 2 seconds before reaching the wheel speed associated with the action primitive derived from the policy.

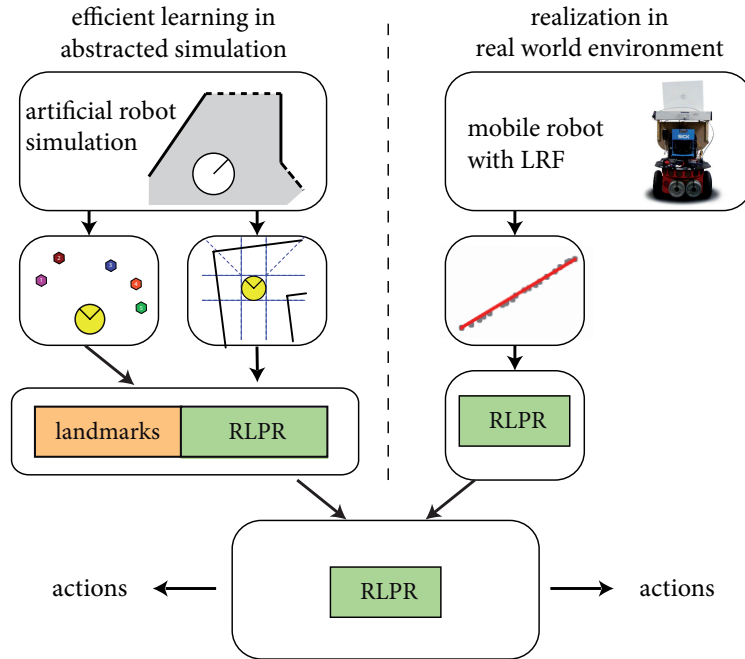


Figure 15: Evolution of spatial representations in both simulation and real robot application. Abstraction enables both scenarios to operate on the RLPR representation to achieve a reasonable action selection.

5.5 Behavior of the Real Robot

To show the utilizability of knowledge transfer to the real robot we derived a structure space policy π_S with APSST from a policy learned in the simulator for 40,000 training episodes as described in Section 5.2.2. This policy could instantaneously be used by the robot control software on the Pioneer-2 as it was generated. The software in both the learning situation and the robot controller used identical code to derive navigation actions.

The office building the Pioneer-2 robot had to cope with shows spatial structures that have not been present in the simulation. For example, neither open spaces (that is: locations where walls are not present in the immediate view) have not been observed in training nor non-rectangular structures like big flower pots that we found in the office floor at the time of the experiment. Also, the rooms were equipped with furniture such as chairs and tables.

While the robot navigated in its environment we have recorded the laser scans to reconstruct the robot trajectory using autonomous mapping techniques.³ With the help of GMapping⁴ (Grisetti, Stachniss, & Burgard, 2007) we obtain the maps and trajectories shown in Figure 16 (b) and (c). The maps result from two individual runs of the robot in the environment shown in Figure 16 (a). In both cases the robot traveled about 50 meter without collisions. All in all we observed the robot to exhibit a reasonable navigation behavior, following corridors in a straight line, crossing open space, and turning smoothly around corners. In particular the robot showed ability to cope with structural

³To support the review process, we also provide a short movie of the robot navigating. It is available for download under <http://www.sfbtr8.spatial-cognition.de/project/r3/media/>

⁴available from <http://www.openslam.org>

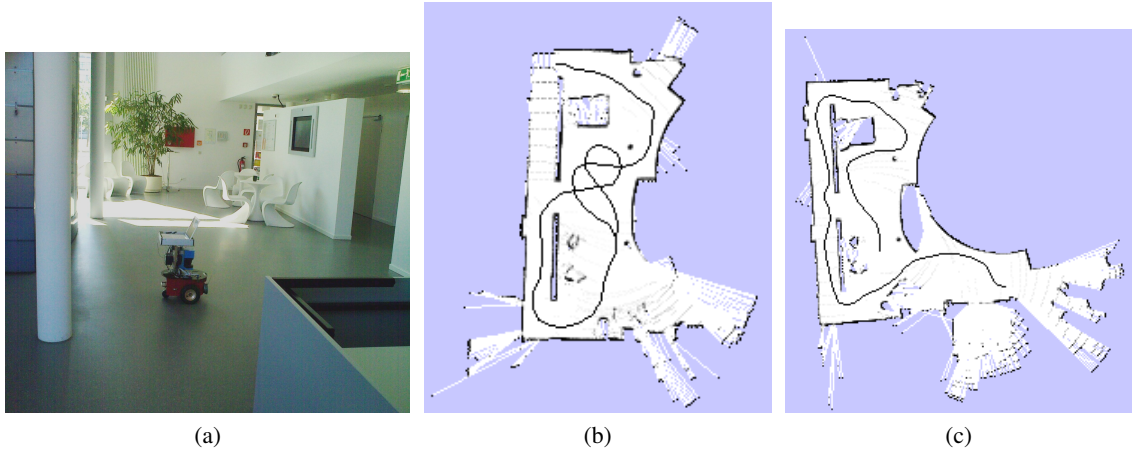


Figure 16: Trajectories of the robot constructed by mapping the environment with a laser scanner. (a) The robot in the test environment. (b) Run with travel distance 53m. (c) Run with travel distance 54m.

elements not present in the simulated environment, such as open space or smaller obstacles: All of those are generalized to line segments and thus being regarded as if they were walls. This also holds for dynamic obstacles as, for example, humans: If they do not move too fast, the robot interprets them as (moving) line segments and is able to drive around them without collisions. All general navigation skills learned in simulation have been transferred to the real-world environment.

The robot only got stuck when reaching areas with a huge amount of clutter (such as hanging leaves of plants) and in dead ends where the available motion primitives do not allow for collision-free movement anymore—driving backward was not available in the set of action primitives. Because the original task was goal-oriented (searching a specific place), the robot also showed a strong tendency of moving forward and thus actively exploring the environment instead of just avoiding obstacles.

6 Summary and Conclusions

This paper presents a study on abstraction with the goal of identifying principles that allow an agent to transfer knowledge from one learning task to another. In particular, we show how abstraction can be used to extract a generally sensible behavior of an agent a learned policy and how it can be utilized for transfer learning.

We distinguish three facets of abstraction: aspectualization, coarsening, and conceptual classification to give crisp definitions of concepts that are often used in a fuzzy manner. We show that different abstraction principles can be employed for obtaining a desired effect; in particular, coarsening and aspectualization are equally expressive. Aspectualizable knowledge representations are the essential prerequisite for our framework to knowledge transfer in reinforcement learning. By employing a knowledge representation that creates a structure space aspectualizable state space it is possible to transfer knowledge learned in context of the source task to the target task. We present the method of a posteriori structure space aspectualization (APSST) to extract generally sensible behavior and show how it can be used for knowledge transfer to new tasks. In an experimental validation

of our approach we demonstrate that navigation skills learned in a dramatically simplified simulator can also be transferred to real robot navigation. The robot is able to adapt to the new scenario as its behavior retains sensible navigation skills from the simulation, as for example, to avoid obstacles and navigate along corridors or turns.

Acknowledgments

This work was supported by the DFG Transregional Collaborative Research Center SFB/TR 8 “Spatial Cognition”. Funding by the German Research Foundation (DFG) is gratefully acknowledged.

References

- Bertel, S., Freksa, C., & Vrachliotis, G. (2004). Aspectualize and conquer in architectural design. In J. S. Gero, B. Tversky, & T. Knight (Eds.), *Visual and spatial reasoning in design III* (p. 255-279). Key Centre of Design, Computing and Cognition, University of Sydney.
- Bittner, T., & Smith, B. (2001). A taxonomy of granular partitions. In *Proceedings of COSIT* (p. 28-43).
- Boyan, J. A., & Moore, A. W. (1995). Generalization in reinforcement learning: Safely approximating the value function. In *Proceedings of NIPS 1994* (pp. 369–376).
- Dean, T., & Givan, R. (1997). Model minimization in markov decision processes. In *Proceedings of AAAI* (p. 106-111).
- Dietterich, T. G. (1998). The MAXQ method for hierarchical reinforcement learning. In *Proceedings of ICML* (pp. 118–126).
- Dietterich, T. G. (2000). State abstraction in MAXQ hierarchical reinforcement learning. In *Proceedings of NIPS 1999* (p. 994-1000).
- Dorigo, M., & Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, 71(2), 321–370.
- Fernández, F., & Veloso, M. (2006). Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of AAMAS* (p. 720-727).
- Frommberger, L. (2006). A qualitative representation of structural spatial knowledge for robot navigation with reinforcement learning. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*.
- Frommberger, L. (2008, June). Learning to behave in space: A qualitative spatial representation for robot navigation with reinforcement learning. *International Journal on Artificial Intelligence Tools*, 17(3), 465-482.
- Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23, 34–46.
- Gutmann, J.-S., Weigel, T., & Nebel, B. (2001). A fast, accurate and robust method for self-localization in polygonal environments using laser range finders. *Advanced Robotics*, 14(8), 651–667.
- Herskovits, A. (1998). Schematization. In P. Olivier & K. P. Gapp (Eds.), *Representation and processing of spatial expressions* (p. 149-162). Lawrence Erlbaum Associates.
- Hobbs, J. R. (1985). Granularity. In *Proceedings of IJCAI* (p. 432-435).
- Jong, N. K., & Stone, P. (2005). State abstraction discovery from irrelevant state variables. In *Proceedings of IJCAI* (pp. 752–757).
- Kaelbling, L. P., Littmann, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.
- Klippel, A., Richter, K.-F., Barkowsky, T., & Freksa, C. (2005). The cognitive reality of schematic maps. In L. Meng, A. Zipf, & T. Reichenbacher (Eds.), *Map-based mobile services – theories, methods and implementations* (p. 57-74). Springer.
- Konidaris, G. D. (2006). A framework for transfer in reinforcement learning. In *Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*.
- Konidaris, G. D., & Barto, A. G. (2006). Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of ICML* (p. 489-49).
- Konidaris, G. D., & Barto, A. G. (2007). Building portable options: Skill transfer in reinforcement learning. In *Proceedings of IJCAI*.

- Li, L., Walsh, T. J., & Littman, M. L. (2006). Towards a unified theory of state abstraction for MDPs. In *Proceedings of the ninth international symposium on artificial intelligence and mathematics* (pp. 531–539).
- Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable markov decision processes. *Annals of Operations Research*, 1, 47-66.
- Mackaness, W. A., & Chaudhry, O. (2008). Generalization and symbolization. In S. Shekhar & H. Xiong (Eds.), *Encyclopedia of GIS* (p. 330-339). Springer.
- McCallum, A. K. (1995). *Reinforcement learning with selective perception and hidden state*. Unpublished doctoral dissertation, University of Rochester.
- Moravec, H. P., & Elfes, A. E. (1985). High resolution maps from wide angle sonar. In *Proceedings of ICRA*.
- Munos, R., & Moore, A. (1999). Variable resolution discretizations for high-accuracy solutions of optimal control problems. In *Proceedings of IJCAI* (p. 1348-1355).
- Parr, R., & Russell, S. J. (1998). Reinforcement learning with hierarchies of machines. In *Proceedings of NIPS 1997* (p. 1043-1049).
- Reynolds, S. I. (2000). Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In *Proceedings of ICML*.
- Schlieder, C. (1993). Representing visible locations for qualitative navigation. In N. P. Carrete & M. G. Singh (Eds.), *Qualitative reasoning and decision technologies* (pp. 523–532).
- Stell, J. G., & Worboys, M. F. (1999). Generalizing graphs using amalgamation and selection. In *Proceedings of SSD* (p. 19-32).
- Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: an introduction*. MIT Press.
- Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2), 181-211.
- Talmy, L. (1983). How language structures space. In H. L. Pick Jr. & L. P. Acredolo (Eds.), *Spatial orientation: Theory, research, and application* (p. 225-282). Plenum.
- Taylor, M., & Stone, P. (2007). Cross-domain transfer for reinforcement learning. In *Proceedings of ICML*.
- Taylor, M., Stone, P., & Liu, Y. (2005). Value functions for rl-based behavior transfer: A comparative study. In *Proceedings of aai* (p. 880-885).
- Thrun, S., & Schwartz, A. (1995). Finding structure in reinforcement learning. In *Proceedings of NIPS 1994*.
- Torrey, L., Shavlik, J., Walker, T., & Maclin, R. (2006). Skill acquisition via transfer learning and advice taking. In *Proceedings of ECML* (p. 425-436).
- Watkins, C. (1989). *Learning from delayed rewards*. Unpublished doctoral dissertation, Cambridge University.
- Zadeh, L. A. (1979). Fuzzy sets and information granularity. In M. M. Gupta, R. K. Ragade, & R. R. Yager (Eds.), *Advances in fuzzy set theory and applications* (p. 3-18). North Holland Publishing Company.