Master's Thesis

Structure Learning for Robotic Motor Control

Graz University of Technology
Faculty of Computer Science
Degree Programme: Telematics

Tim Genewein

Graz, November 2011

Supervisor: O.Univ.-Prof. Dipl.-Ing. Dr.rer.nat Maass Wolfgang
Institute: Institute for Theoretical Computer Science

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

(date)	(signature)

Foreword

This work is the result of a Master's thesis, conducted from April to November 2011 at the *Institute for Theoretical Computer Science* (IGI), Graz University of Technology. Besides a general interest in artificial intelligence and robotics, the lectures of *Wolfgang Maass* on machine learning have sparked my interest in Bayesian networks and led to the decision to devote my Master's thesis to a related research question. Maass, Neumann and Rückert provided a quite detailed and elaborate proposal that formed the basis of this work. I want to thank Wolfgang Maass for his pragmatic supervision and the possibility to use the institute's infrastructure and work environment. A special thanks goes to *Gerhard Neumann* and *Elmar Rückert* for constantly providing impulses and their support on difficult tasks of the thesis.

To me personally, the work on this thesis was a very valuable experience - not only was I able to familiarize myself with an interesting topic of research, but I could also get to know the scientific methodology while working on a challenging project.

My final thanks goes to my family and all those who have been supportive to enable my studies.

Tim Genewein

Abstract

One crucial aspect of human and animal learning is the ability to gain *abstract knowledge* about a certain class of (similar) problems. The process of generalization can then be interpreted as applying the abstract knowledge to novel problems. Experimental studies have shown that in many natural tasks humans and animals are able to learn the *structure* of a task and furthermore exploit this knowledge when facing a new but similar task. The structure can be seen as a "general set of rules" or structural invariants of the parameters of a problem-class. In this thesis, a hierarchical Bayesian network is used to perform *structural learning* on a robotic problem-task, where a robotic agent tries to move towards a goal-position. It does so by issuing movement-commands, i.e. setting its velocity. The actual movement is perturbed with a rotation-distortion of an unknown angle. The idea is to learn the rotation-distortion as the structure of the problem and when being confronted with a novel rotation-angle exploit the already gained structural knowledge to be able to rapidly adapt to the novel rotation-distortion and finally compensate it.

This thesis evaluates the qualities of the proposed model in terms of structural learning and the ability to rapidly adapt to novel but similar problems. The thesis shows that the proposed model is actually able to extract the structure of the task and provide general knowledge that can be exploited when adapting to novel instances of the task, with a new rotation angle. The results also show that the number of (time-) steps required for learning can be significantly reduced by using the abstract prior-knowledge.

The thesis also highlights some shortcomings of the proposed model and used methods. The most severe drawback is that the number of prior mixture-components is unknown and can not be inferred from the given data. Furthermore, the model has the flavor of being tailored for the given class of problem tasks, which contradicts the idea of a system that is able to *truly* adapt and generalize. Potential future research could be aimed towards a similar model that is able to cope with continuous environments.

The approach shown in this thesis is novel and has not been presented before. The results highlight the capabilities of rapidly adapting to novel but similar tasks by exploiting abstract knowledge, which has been gained through **structural learning**.

Kurzfassung

Ein zentraler Aspekt von Lernprozessen in Mensch und Tier ist die Fähigkeit sich *abstraktes Wissen* über eine Reihe von ähnlichen Problemen anzueignen. Generalisierung kann dann als die Anwendung dieses abstrakten Wissens auf neue Probleme angesehen werden. Experimentelle Studien am Menschen, sowie zahlreiche Tierversuche haben gezeigt, dass die Probanden in der Lage waren die *Struktur* einer Problemstellung zu lernen und dieses Wissen dann für das Erlernen eines neuen, aber ähnlichen, Problems nutzen konnten. *Struktur* ist in diesem Kontext als strukturelle Invarianten der Parameter oder eine Art "generelles Regelwerk" der Problemstellung anzusehen. In dieser Arbeit wird ein hierarchisches Bayes'sches Netz dazu verwendet, die Struktur eines Steuerungsproblems aus der Robotik zu lernen. Es handelt sich dabei um einen Agenten, der versucht eine geweisse Zielposition anzufahren. Zu diesem Zweck kann der Agent Steuerkommandos vorgeben, mit denen er seine Geschwindigkeit (in beiden Achsen) setzen kann. Die tatsächliche Bewegung des Roboters ist jedoch mit einer Rotation unbekannten Winkels überlagert. Die Kernidee ist es, diese Störung als die Struktur des Problems zu lernen und dieses Wissen einzusetzen, wenn der Agent mit einem neuen, bislang unbekannten, Rotations-Winkel konfrontiert wird. Mit Hilfe des strukturellen Wissens soll der Agent in der Lage sein sich in kürzester Zeit an die neue Störung anzupassen und diese zu kompensieren.

Diese Arbeit evaluiert die Eigenschaften des vorgeschlagenen Modells im Hinblick auf strukturelles Lernen und die Fähigkeit zur raschen Anpassung an neue, aber ähnliche, Problemstellungen. Die Arbeit zeigt, dass das Modell tatsächlich in der Lage ist, die Struktur der Aufgabe zu extrahieren. Darüber hinaus kann damit abstraktes Wissen erzeugt werden, das beim Lösen eines ähnlichen Problems (Rotation mit bisher unbekanntem Winkel) eingesetzt werden kann. Außerdem zeigen die Ergebnisse dieser Arbeit dass die Anzahl an (Zeit-) Schritten, die für die Anpassung an die neue Aufgabe benötigt werden, durch strukturelles (Vor-) Wissen signifikant reduziert werden kann.

Die Arbeit zeigt auch gewisse Mängel des vorgeschlagenen Modells sowie der verwendeten Methoden auf. Am schwersten wiegt der Umstand, dass die Anzahl an Mixture-Komponenten unbekannt ist und auch nicht aus den Daten abgeleitet werden kann. Außerdem entsteht der Eindruck, dass das Modell auf die gegebene Aufgabenstellung "maßgeschneidert" ist - was dem Konzept eines generalisierenden Systems widerspricht. Weiterführende Arbeit könnte auf ein Modell gerichtet werden, das in kontinuierlichen Problemstellungen eingesetzt werden kann.

Der Ansatz, der in dieser Arbeit verfolgt wird, wurde in dieser Form noch nicht gezeigt. Die Ergebnisse unterstreichen die Fähigkeit zur raschen Adaption an neue, aber ähnliche Problemstellungen unter der Ausnützung von abstraktem Wissen, das durch Methoden des **Strukturellen Lernenes** gewonnen wurde.

Contents

List of Figures

1 Introduction

"One reason that animals and humans can rapidly learn new problems is perhaps because they take advantage of the high degree of structure of natural tasks."

S.J. Gershman, Y. Niv [1]

1.1 Motivation

Consider a robot that has learned to play tennis. It is able to return an incoming ball by hitting it the right way with a tennis racket. This requires some kind of sensory system to determine the trajectory of the incoming ball and an actuator to hit the ball with the racket. The learning problem then consists of finding a certainly nonlinear *mapping* between the sensory input and the motor output. A lot of research in the area of robotics has been targeted at finding efficient ways to build such mappings. While several elegant and powerful approaches are known, this problem is still a topic of ongoing research.

Now consider the adaption of the robotic system to play squash or table tennis. Due to the different physical properties of the new ball and racket, the sensorimotor mapping has to be different. Intuitively, one could argue that the new mapping will probably be quite *similar* to the tennis-mapping. Or more generally speaking, that the mappings for different racket sports will be "more similar" than those for example for soccer and basketball. It is reasonable to postulate, that all possible racket sports mappings share some common structural features, e.g. certain parameter covariances. If that postulate holds, the learning problem could be reformulated. Instead of learning a sensorimotor input-output mapping only, it is desirable to simultaneously extract knowledge about the structure of all mappings for a range of similar tasks - or in other words: to *learn the structure of a task*.

Under the assumption that natural tasks that are related, show a certain, non-trivial structure, the problem of learning such tasks decomposes into two learning aspects:

Structural learning: The extraction of abstract knowledge (=structure) out of different instances of the same task with small variations. This can usually also be done simultaneously to parametric learning.

Parametric learning: The discovery of optimal parameters for a single instance of the task, assuming a known/fixed task structure, also known as policy learning.

This terminology is used in [2, 3], which give additional illustrative examples on structural learning and also emphasize the importance of task variation. The terms originally arise from the field of *Adaptive Control* [4].

The question, whether the initial assumption holds or not, can not be easily answered, since the assumption statement does not give a sharp definition. However from a more practical point of view there are results that suggest the existence of significant structures in a lot of natural tasks. Kemp and Tenenbaum have been quite active in this area (see e.g. [5, 6]). Braun et. al. also provide a nice discussion with a lot of further references in [3].

Braun et. al. have been able to find and quantify structural learning processes in a series of experiments with human test subjects [2, 3, 7] - where they have mainly considered sensorimotor learning. Furthermore they highlight three key features of structural learning in sensorimotor tasks - this work is related to the first aspect:

- *"Facilitation of learning tasks with the same structure"*
- *"Reduced interference when switching between tasks that require opposite control strategies"*
- *"Preferential exploration along the learned structure"*

quoted from [2]

Additionally structural learning can also lead to a dimensionality reduction of the parameter search-space. Gershman, Niv provide a very good introduction and overview of current research in [1] where they also provide more insight on the dimensionality reduction with respect to reinforcement learning. A more in-depth discussion of structural learning can be found in 2.1.

One problem in today's robotics is the lacking ability to rapidly adapt to novel but related situations. What seems natural to humans, turned out to be a severe challenge for artificial systems. The concept of structural learning provides new insight to the ability of acquiring abstract knowledge out of concrete instances of a task and it has already lead to new models in machine learning that are capable of doing so. These models have caught the attention of the machine learning community but since the theoretical foundations are not too extensive yet, applications can only be found on minimalistic problems [7, 26, 5].

The motivation for this thesis originates from the promising benefits of structural learning and the idea to use hierarchical Bayesian models to exploit them; resulting in powerful models that are able to **rapidly** adapt to new but similar situations, as well as to produce transferable knowledge that allows to exploit gained experience when learning a new task. A brief introduction to Bayesian models can be found in 2.2.

A remark for readers that are already familiar with Bayesian networks: methods for learning the topological (graph-) structure of a Bayesian network are also known by the term "structure learning" - in order to avoid any misconceptions: this thesis is not concerned with topological structure learning and will use the terms *structural learning* and *structure learning* synonymously to refer to the process of extracting structural invariants from a set of related tasks.

1.2 Problem-task setup

In order to investigate the qualities of the proposed models, they will be evaluated on a simple problem, inspired by the experiments of Braun et. al. [2]. The simulation setup consists of a (robotic) agent, trying to navigate towards a certain position in a two-dimensional plane. In order to reach that goal, the agent can set its velocity on the x- and y-axis. However, the movement of the agent is **superimposed** with a rotation that is unknown to the agent - e.g. if the agent tries to increase its x-velocity only, but the rotation has an angle of 90°, the command will actually affect the agent's y-velocity. To make the learning problem more challenging, the agent's commands and its movement in the 2D-plane are noisy. Figure 1.1 shows a schematic depiction of the problem-task.

The angle of the rotation is kept constant throughout the duration of a single task rollout but varies among different instances of the task. Since the angle is unknown to the agent, this makes the *rotation-angle a latent variable of the problem-task.* The goal of the parametric learning problem is then to find a mapping between the current position and velocity of the agent (= sensory input) and a corresponding velocity-command (= motor output) in order to reach the goal-position. Simultaneously the agent should gain abstract, structural knowledge that can be transferred to a novel task instance where it should simplify the parametric learning process. E.g. if the agent has already learned a mapping for a rotation of 35°, the mapping for a rotation of 42° will be very similar. Here the "structural" knowledge is related to the similarity of mappings for close-by rotation angles. This similarity is only affected by the superimposed movement distortion - in this case the rotation, but one could also imagine other distortions, like a shearing or a scaling.

The structure of the problem-task is governed by latent variables of the task - in this case the structure is a rotation-distortion that depends on the rotation angle.

Note that there are no strong assumptions on the structure - if it was known that there is always a rotation-distortion, the agent could simply perform a single movement step and then compute the rotation angle - the whole problem would become trivial. Instead, the model should implicitly capture the structure of the problem setup and it must be able to cope with various kinds of movement distortions. The model should produce generalized, transferable knowledge!

Considering a discrete-time environment, the agent has to plan several subsequent steps to reach the goal-position. If the agent is able to directly set its velocity, the planning problem is trivial, because at any concrete time-step it is sufficient to consider the immediate next step only. In order to make the planning non-trivial, the problem task can be extended to use *accelerations as actions*, instead of setting velocities directly, which requires "planning a few steps ahead".

1.3 Research objectives

Recently, there have been quite a few approaches starting from different backgrounds but working towards a common problem. Depending on the initial background, the terminology might

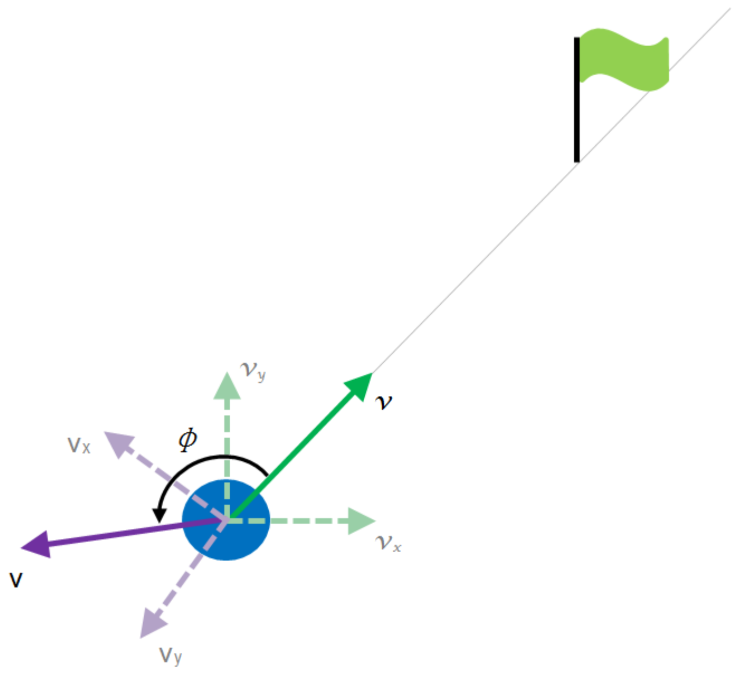


Figure 1.1: Problem-task - the agent (blue circle) tries to move towards the goal position (green flag). The corresponding velocity-commands are depicted in green (cursive captions). Due to the rotation distortion with angle ϕ, the actual velocity (purple, sans-serif captions) will point towards a completely different direction.

be different (structural learning, learning to learn, learning on several layers of abstraction, ...) and the research question might have a bit of a different flavor. However, it seems as if this work is starting to emerge into a new branch of research which might help in understanding learning in general as well as learning processes in humans and animals but also provide inspiration for novel models and approaches, not only in robotics.

The *Institute for Theoretical Computer Science* (IGI) at Graz University of Technology has a general interest in the topic but also a specific interest in applications of structural learning in the field of robotics. The topic for this thesis, including the problem-task as well as the model, capable of extracting structural invariants of the task, have been proposed by the head of the institute (Maass W.) and two members of the robotics group (Neumann G., Rückert E.).

This thesis contributes towards the ongoing research (at the institute) with:

- An **implementation** of a simulation environment to investigate the qualities of the proposed model using the proposed task, inspired by the work of Braun et. al.
- The planning and **execution of experiments** within the simulation environment to gain both, qualitative insights as well as quantitative results
- A **critical reflection** on the proposed approach in consideration of the simulation experiments. Of special interest are the capabilities for extracting abstract knowledge and using it to rapidly adapt to a novel but similar task.

Large parts of the mathematical derivations as well as improvements to the originally proposed model have been developed with Neumann, Rückert and also many ideas for experiments have either been proposed by or worked out in cooperation with the group and thus are not *solely* the work of the author. All implementations, simulations and the corresponding results are, however.

The goal of this thesis is to *demonstrate* the application of the proposed model on the given problem task and show its potential strengths and shortcomings. It can be seen as some kind of "proof-of-concept" or perhaps even groundwork for more complex problems. Furthermore it provided a very suitable problem for the author to familiarize with the concepts of structural learning and gain hands-on experience in the field. It is not within the scope of this thesis to gain vast theoretical insights on generalization and abstraction or to evaluate the biological/neuroscientific relevance of the model.

1.4 Thesis structure

The thesis is organized as follows:

Chapter 2 presents fundamental background information on the topics of structural learning and Bayesian models. It highlights the key-concepts and provides references for further reading, rather than an extensive coverage of the topics.

Chapter 3 describes the problem-task in detail as well as the hierarchical Bayesian model used for structural learning. It provides insight on the methods used to obtain the results presented in this thesis. Furthermore it describes an advanced problem task setup as well as some extensions to the basic Bayesian model. The chapter also provides (mathematical) details on the process of fitting the model parameters to observation data (using expectation maximization).

Chapter 4 shows the results of various experiments performed in the implemented simulation environment. Most results are highlighted with illustrative plots and corresponding interpretations.

Chapter 5 provides a critical reflection on the results as well as a discussion on some of the shortcomings of the model as well as the current implementation.

The Appendix gives additional details on parts of chapter 3 which are not crucial in understanding the concepts but required for following the details of the mathematical results. It also provides some background on the planning algorithm (value iteration) which is not within the main-scope of this thesis, yet it is too important to simply be omitted.

A lot of the plots in this thesis use a colormap for depicting values - some figures also show the plots of several graphs, distinguishable via different colors. These plots might be ambiguous in a monochrome print and the reader is politely referred to the digital version of this thesis, if a coloured print is not available.

2 Fundamentals

The previous chapter has introduced a problem task where a robotic agent should learn to compensate rotation-distortions of various angles. The rotation-distortion was mentioned as the *structure* of the task. This chapter starts with *a closer look on structural learning* to shed some light on the term *structure* in the context of task-invariants, abstract knowledge or a "general set of rules" that govern the task. Perhaps the attributes of structural learning become more clear in contrast to parametric learning. The section then continues with a brief overview of some of the related fields and references interesting experimental results. At the end of the section one particular experimental study by Braun et. al. is presented in detail. The experimental setup of this study was the main-inspiration for the problem task of this thesis.

The second section gives a very brief introduction to Bayesian networks and methods for inference and learning (i.e. parameter-fitting and topological structure learning) of such models. As the topic is vast and extensive, only some of the key-concepts can be mentioned. However, the section references several sources that provide in-depth information on the topics covered. Since the models used in this thesis are *hierarchical Bayesian models* as well, this section provides some background information and starting-points for further literature research for readers that are unfamiliar with Bayesian networks.

This chapter provides basic information on the most important aspects of this thesis. It does not contain specific information on the methods and models used (as the next chapter does). Readers, who are already familiar with the concepts presented in this chapter are welcome to simply skip it.

2.1 A closer look on structural learning

The initial, motivating example in Sec. 1.1 introduced a (fictional) robotic task, where the goal was to teach a robot to play tennis. By a slight variation of the task-setup (table-tennis), an intuitive motivation for structural learning processes was given: the naturally arising intuition that the control parameters for both of these racket sports must be somewhat similar or share a certain *structure*. The intuition that similar tasks actually share certain structural features and that these structural invariants play a substantial role when learning such tasks was proven to be correct in quite a number of experimental studies (see 2.1.1).

To get more insight on *the structure of a task*, reconsider the motivating example. Assume that the robotic system has learned to play tennis (regardless of how it did so). The dozens or perhaps hundreds of control-parameters have been set to values that lead to the desired results. In order to be able to play table tennis, the system has to find the corresponding parameter settings. Take a look at Figure 2.1 and assume that the red-dot indicates the parameter values for tennis (which are known) and the blue dot indicates the (unknown) values for playing table tennis. To be able to show simple plots, only two (out of possibly hundreds) of the parameters are considered - each one corresponding to one axis.

Without any additional knowledge, it would be necessary to search the whole parameter-space, more or less randomly, in order to obtain the blue-dot solution. However, it seems natural to assume that most of these parameter settings will not yield reasonable solutions. And further, one could assume that the parameter values have to satisfy certain covariances. In the figure this is indicated by the thick, black line - all solutions for different, but similar, task-setups lie on this line. Now, if this line was known - the search for a new-parameter set could significantly be simplified, because one would only have to search along that line. This effectively means a dimensionality-reduction since it is no longer necessary to search the whole two-dimensional parameter space but simply along a one-dimensional meta-parameter μ. The meta-parameter governs the transition from one parameter-set to another. This is also depicted in Figure 2.1(B).

If the problem shows structural invariants, they correspond to the “black line” or the meta-parameter μ and the task of **structural learning** is to find these structural invariants; which probably requires that the system has observed many different task-instances. Once the meta-parameters have been found, adapting to novel task-instances that share the same underlying structure, becomes easier (parametric learning), because exploration can be guided by the (lower-dimensional) structure. Especially for high-dimensional (optimization-) problems, the computational demand could significantly be simplified if the problem has a structure that lies on a lower-dimensional manifold. Even, if the desired solution does not exactly lie on the learned structure (green dot) - the initial exploration could still follow the learned structure.

The meta-parameter μ that captures the structure of a problem, could also be seen as a latent-

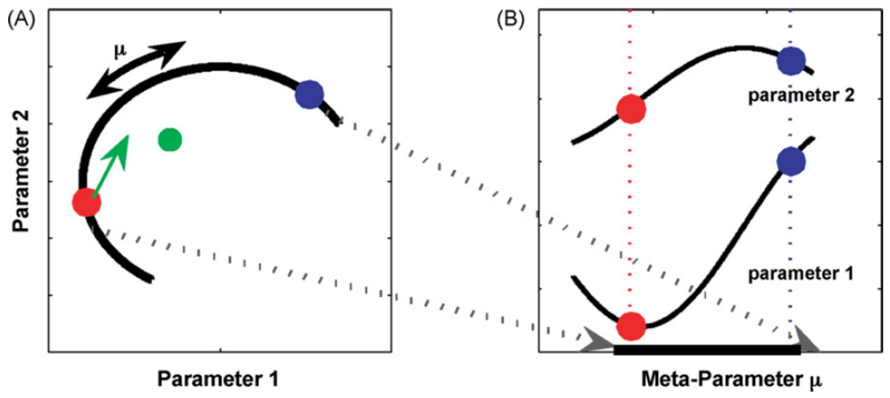


Figure 2.1: Schematic sketch of structural learning - the transition between the sets of optimal parameter values (red and blue) for two similar tasks is governed by a meta-parameter μ. This meta-parameter can be modeled as a latent variable and corresponds to the *structure* of the problem. Once the structure has been learned, finding optimal parameters for novel instances of the same problem-class can be simplified, because it is no longer required to search the entire parameter-space. Instead, the search can be restricted to follow the learned meta-parameter, which in this case results in a dimensionality reduction from two to one dimension. Figure originally from [2] and included with permission.

variable of the problem-task; which already provides inspiration for modelling the structure with probabilistic models.

2.1.1 Related fields

Structural learning effects have been observed in many studies, mostly in the field of animal or human cognition, but also in other areas such as sensorimotor-learning. However, not always has the main focus been on structural learning - a lot of experimental studies were dealing with discrimination tasks (where the test-subject learns to discriminate objects based on a certain feature such as its color or shape). Many other studies investigated the subject's ability to learn causal relationships but not the speedup when learning novel, but similar tasks. In both kinds of studies the focus was usually not explicitly set on structural learning but of course the results are particularly interesting under this point of view.

H.F. Harlow conducted experiments, where monkeys had to choose between one of two stimulus-objects and only one object would lead to a reward. During a single block the same pair of objects was presented at different locations and for different blocks, different pairs of stimulus-objects were used. Harlow could show that the monkeys were able to learn faster in the later trial-blocks, compared to the initial blocks where they would learn quite slowly. This suggests that the monkeys were able to learn the structure of the problem (i.e. one of two objects leads to a certain reward). Harlow was also one of the first to notice that task-variation is crucial for learning the structure and actually most studies focus on a single problem-task without any variations (which does not allow the extraction of much structural knowledge). Harlow presented his theory of "learning to learn" or meta-learning and the formation of so called "learning-sets", which formed the basis for a new point of view on cognitive studies as well as learning processes in [8]. In a more recent work Reznikova provides lots of examples and insights to animal behavior and cognition from various points of view - he also presents results where monkeys were trained to learn quite complex rules, such as discriminating based on shape, if the object is presented on a white background and discriminating based on colour, if the background is black (see [9]).

There is a vast number of cognitive experiments with humans where the learning-to-learn hypothesis could be applied. However, most of these results do not consider the (potential) speedup for learning novel tasks of the same structure. Braun et. al. conducted visuomotor experiments, focussing exactly on the aspect of increased learning performance when presented with similar tasks. The experiments are presented in detail in the next Section 2.1.2. Kemp, Tenenbaum present in [10] quite a few examples of how structured background knowledge influences human reasoning and they also show (Bayesian) models that are capable of expressing background knowledge over a number of structures. In [11], Turnham et. al. observe structure in covariances of visuomotor priors in humans. The authors of [12] present the idea that seemingly suboptimal human choices in certain experiments, might be optimal under different conditions. They propose that humans will simultaneously learn the structure of reward generation as well as the environment (and e.g. assume changes in the environment that are not reflected by the experimental setup) and thus their choices can not be compared against mathematical models that do not incorporate these learning processes.

Among these experimental results that suggest the presence of certain structural learning processes in animals and humans, there are also some theoretical advantages or benefits of structural learning. One very promising potential benefit is a (strong) *dimensionality reduction*. As far as implementations are concerned (even biological implementations) this reduction in the size of the search space will always lead to reduced computational demands. For some methods, such as reinforcement learning, such dimensionality reductions are highly desired.

On the other hand, structural learning can also enable fast-learning (not only through dimensionality reduction and the reduced computational cost). Once the structure of a task has been learned, it is represented as abstract *transferable* knowledge that can be used for novel tasks. The key-element here is the extraction of generalized knowledge from concrete task-instances. It is no longer required to learn a similar task from scratch, but the structural knowledge can be used to initialize parameter-values and guide the exploration of these parameters during optimization. From a conceptual point of view, this ability to generalize is very appealing as it addresses one of the key-problems in today's research. Especially in robotics, it could help in creating systems that are actually able to "leave the lab" because they can *rapidly* adapt to their environment. In today's machine learning, even mild task-variations usually require huge adaptation processes, mostly because current models do not incorporate such task-variations (because it would quickly lead to computationally infeasible models) and if they do, they mostly do not use representations where these variations lead to generalized knowledge.

Optimal control theory provides a theoretical background for solving (the most) complex problems. But for these approaches and methods based on it, computational demand explodes with the dimensionality and size of parameter-search-spaces. Bellman and his work on optimality is closely related and suffers from the same problems (see also A.2). Methods based on these approaches have an inherent need for dimensionality reduction and could potentially benefit significantly from structural learning.

In the field of (computational) neuroscience, structural learning could also provide new insights on neural learning processes. Consider the vast amount of sensory input that reaches the cortex - for many tasks, such as reacting to a stimulus on your finger-tip or a visual stimulus in a small part of the visual field, only a very small subset of the current sensory-input is relevant. While it is known that the Thalamus as well as the mechanism of attention play an important role in controlling the sensory input, it is reasonable to argue that there must also some kind of structural mechanism be involved. For many tasks, the relevant sensory input has to be learned and this learning process could correspond to some kind of topological structural learning. The Hebbian rule ("what fires together wires together") would not necessarily require such structural processes - but when thinking about complex cognitive tasks that rely on, say a visual input, it seems plausible to assume such processes that would also lead to a significant dimensionality reduction. But also when adjusting synaptic parameters (probably hundreds of thousands or millions), a dimensionality reduction through abstract, structural knowledge seems reasonable. And while it is hard to find concrete mechanisms, the idea of structural learning has certainly led to new points of view and has also inspired researchers to work towards certain directions.

A short discussion on structure in perception as well as in action and an excellent overview of relevant literature for the emerging field of structure learning can be found in Gershman, Niv

([1]). Their main focus is set on structural learning with respect to reinforcement learning and thus the main-interest lies on the dimensionality-reduction properties.

Currently there exists little theoretical work on how structural learning mechanisms could be formalized, or incorporated into other methods. However, there is a lot of evidence that a many natural tasks show certain structural invariants and that animals as well as humans exploit these structures when learning - especially for generalization and abstraction.

2.1.2 Experimental study by Braun et. al.

Braun et. al. have conducted several experimental studies with humans to investigate structural learning processes and the speedup when learning similar task-instances in [2, 3, 7]. This section presents one of their experiments in detail, since it provides an intuitive example for structural learning but it has also served as one of the main sources of inspiration for the problem-task handled in this thesis.

Consider the following experimental setup: The test-subject is in a virtual reality environment that precisely tracks the position of the subject's hand - but the subject is not able to directly see its hand but only a projection of it in the virtual reality. Furthermore, the subject is presented a virtual target that it should reach with its hand - feedback is only provided via the virtual hand. The core-problem of the task is that the movement of the virtual hand is overlaid with a certain distortion, i.e. the virtual hand does not move exactly the same way as the hand of the subject does. The distortion is some kind of linear transformation.

As usual for an experimental study, the subjects have been divided into several groups - where every group experienced 800 trials:

Rotation group: These test-subjects only experienced rotation-distortions, but the rotation-angle was changed every eighth iteration. The mean-rotation angle over all trials was zero.

Random group: This group experienced a random linear transformation (rotation, shearing, scaling) but it received the same amount of $\pm 60^\circ$ rotations (without any other transformations) as the rotation-group.

Control group: This group did not experience any distortions at all.

After this initial phase, all three groups were confronted with a block of 50 rotation-trials with a $+60^\circ$ rotation-angle. The immediately following block (50 trials) then had a rotation-angle of -60°, which requires the exact opposite control strategy. The final block had again an angle of $+60^\circ$.

Braun et. al. intended to show that the subjects (in the rotation group) were able to learn abstract knowledge on the structure of the task to disprove the common opinion that task-variations would prevent learning but instead lead to averaging effects (the average angle for the rot-group was 0°, if the assumption was correct then the rot-group should show an equal performance as the control-group).

The striking results of Braun et. al. were that the rotation-group would perform a lot better than the random- and the control-group (see [2] for the full paper). To show this, several error-measures were used, such as the initial error to the target (200ms after onset of the movement), the cumulative error to the target but also the trajectories itself as well as the average speed during movement.

Figure 2.2 shows the results for the initial angular error. The first plot (A) illustrates the increased learning performance of the rotation-group - the group is able to faster adapt to the novel task, compared to both other groups. It shows a strong facilitation when learning tasks with the same structure. When switching to a task that requires an opposite control strategy, test subjects usually have problems with adapting to such a task - the learning performance is significantly decreased. Plot (B) shows the result of such a setup - all three groups have a decreased learning performance, but the rotation-group (which has learned the structure of the task) shows a strongly decreased interference. Even in another subsequent $+60^\circ$ block (C), the rotation group still shows an increased performance over the first 10 trials.

Summarizing the results, the rotation-group has been able to gain structural knowledge in the initial 800-trial learning phase, whereas the random-group has either not been able to extract a lot of structure or simply different structural knowledge that matches their observations (random linear transformations instead of rotations as the structure of their task). Eventhough the random group has seen the same number of $\pm 60^\circ$ rotation trials (without any other distortions) during the course of the 800-trial learning phase. In subsequent evaluation-blocks the rotation group has shown:

- Strong facilitation when learning tasks with the same structure.
- Reduced interference when switching to a task that requires the opposite control strategy but shares the same structure.

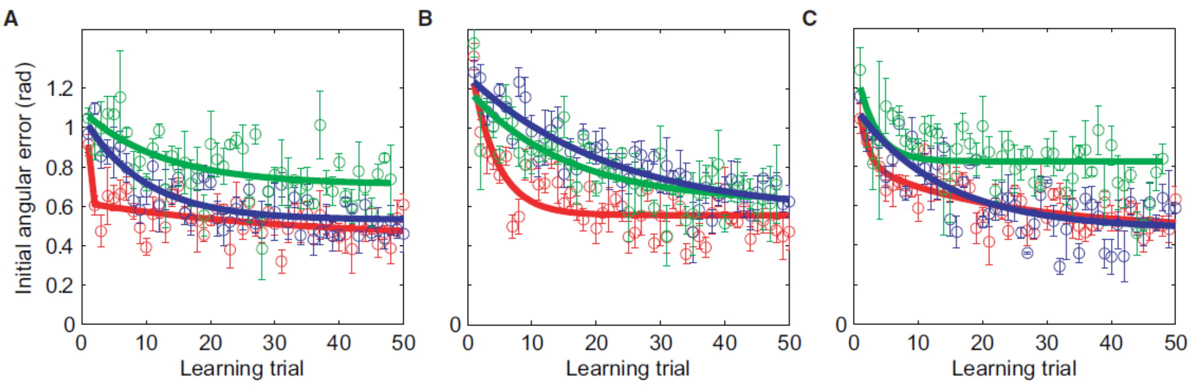


Figure 2.2: Initial angular error - 200ms after the onset of the movement. Rotation-group in red, control-group in blue and random-group in green. (A) Block of 50 trials with $+60^\circ$ - the rotation-group shows a strong facilitation for learning a novel but similar task. (B) Subsequent block with -60° - the rotation-group shows a reduced interference when switching to a task that requires the opposite control strategy. (C) Subsequent block with $+60^\circ$ rotations - the rotation group still shows a performance advantage for the first 10 trials. Figure originally from [2] and included with permission.

Braun et. al. were also able to show that the learned structure will lead to a *preferential exploration* along that structure - these results can be found in [2].

The results allow the conclusion that task-variation does not prevent learning and it will not necessarily lead to averaging; otherwise the rotation-group must have shown the same performance as the control-group, since the mean-angle of the rotation-distortions was zero degrees. In fact, Braun et. al. conclude that task variation is **essential** for structural learning. However, many cognitive experiments do not consider sets of similar tasks, but only a single instance of a task and thus might miss the impact of structural learning effects.

In series of similar experiments, with more or less the same setup, Braun et. al. ([3]) also show the performance-increase over time. Especially interesting are the results on feedforward- versus feedback-control. The feedforward stage refers to the initial phase of movement (the first 300ms) where the human test-subject does not receive any sensory (visual) feedback - because the visual information requires a certain amount of time to be processed. After this feedback comes in, the subject starts to adapt its movement according to the given feedback. This essentially results in two peaks in the mean speed-profiles shown in Figure 2.3. The figure shows the mean speed profiles during subsequent batches of trials - each batch consited of 200 trials. Blueish colors indicate early batches, greenish lines correspond to intermediate batches and the late batches are colored reddish. The subjects had to reach targets in a virtual reality environment, while their perceived movement was overlaid with a rotation distortion.

Taking a closer look, one can see, that the initial (feed-forwad) movement-speed does not increase too much, but the “second peak”, i.e. the movement-speed in feedback-phase increases significantly during learning. Intuitively, this can be explained by the test-subjects being insecure, once they realize that the initial movement went towards the wrong direction. In later stages of the experiment, the subjects have already learned the structure of the task and were able to compensate the rotation-distortion while maintaining quite a high movement speed.

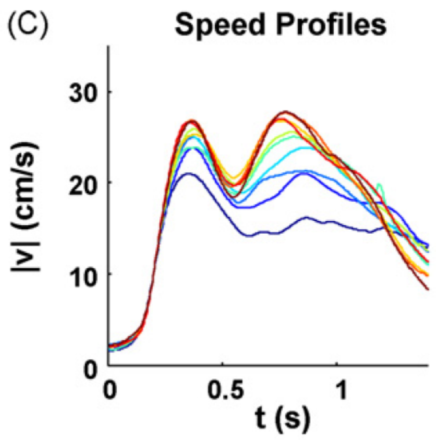


Figure 2.3: Mean speed profiles - each line corresponds to the mean-speed profile of a single batch, consisting of 200 individual trials. Blue indicates early batches, green denotes intermediate batches and late batches are colored red. With an increased number of trials experienced, the test-subjects managed to maintain a high speed in the feedback-control phase (the second peak), because they had learned the structure of the problem and thus were able to compensate rotation-distortions of any angle. Figure originally from [3] and included with permission. The enumerator (C) was kept to be consistent with the original publication.

2.2 Bayesian models

When dealing with real-world information, the problem of inherent *uncertainty* is inevitable. In some cases, the uncertainty might be almost neglectable, e.g. when claiming that the sun will "still be out there" tomorrow. In other cases, uncertainty is a substantial part of the information, e.g. when tossing a coin and predicting the outcome. Thus, the inherent requirement for any technical system that deals with such information is the ability to cope with uncertainty. Furthermore, such systems often suffer from incomplete data. And finally it is often desired to incorporate some kind of prior beliefs. The Bayesian *point of view* addresses these issues very elegantly. At its very core, it is an elegant interpretation of Bayes' theorem:

$$\textit{Posterior probability} \propto \textit{Likelihood (of data)} \cdot \textit{Prior-belief}$$

This "point of view" allows the incorporation of prior beliefs into given data (observations). The big advantage is that both components can be expressed in probabilistic terms (probability distributions) which are inherently capable of handling and even modelling uncertainty. Probabilistic data-models are also very useful when dealing with incomplete data.

Models based on this principle (which can be applied to almost any probabilistic model) are called Bayesian models. However, this is more a term describing certain qualities, rather than a sharp definition. The models can also represented graphically (see Sec. 2.2.1), which then either leads to *Bayesian networks* or *Markov random fields.*

Koller, Friedman discuss in [13] another significant advantage - the so called declarative representation, which refers to the separation of knowledge (in a *model*) and reasoning (in form of *algorithms* that can answer questions based on the model).

"The key property of a declarative representation is the separation of knowledge and reasoning. The representation has its own clear semantics, separate from the algorithms that one can apply to it. Thus, we can develop a general suite of algorithms that apply any model within a broad class, whether in the domain of medical diagnosis or speech recognition. Conversely, we can improve our model for a specific application domain without having to modify our reasoning algorithms constantly."

quoted from [13].

2.2.1 Bayesian networks

The term *Bayesian network* (BN) refers to a graphical representation of a Bayesian model and basically describes *conditional independence statements* of a joint-distribution. Any joint-distribution can always be factorized by applying the product rule (see also [14]) into a product of conditional probabilities. This decomposition becomes especially useful, when some of the involved random variables are conditionally independent of others. In such a case (that is very common for problems of practical interest), many terms of the decomposition vanish or can at least be simplified (e.g. by a smaller number of variables that another variable is dependend on). In fact, this simplification actually enables the application of probabilistic models and techniques for many problems - because otherwise the computational demands would be infeasible.

In [15], a tutorial on learning with Bayesian networks is presented and the introductory section names four significant advantages of Bayesian networks:

- Bayesian methods can readily handle incomplete datasets.
- Bayesian networks allow to learn about causal relationships. This can help in understanding a problem domain, but it also allows predictions based on these causal relationships, even if there is no specific experiment available.
- Bayesian networks facilitate the combination of domain knowledge and (observation) data. In a Bayesian sense, the data (to be precise, the likelihood) and prior-knowledge can be elegantly expressed with probability-distributions and further they can be naturally combined to form a posterior-distribution.
- Bayesian methods provide efficient approaches for avoiding over-fitting to data.

Bayesian networks can be used to capture knowledge. On one hand this can be experimental knowledge or simply a number of observations, but the topological structure of a BN also provides information about causal relationships and conditional independencies of variables which can be seen as domain- or expert-knowledge. Due to this, Bayesian networks in conjunction with the right methods can be used to answer certain questions, based on the model while observed evidence (in the context of these questions) can easily be integrated. This process is referred to as *inference*. On the other hand, Bayesian networks can also be used to gain further insights in the problem domain because they model causal relationships and independence properties as well.

Bayesian networks are also known under the names: (probabilistic) belief network (PBN), probabilistic causal network or directed acyclic graphical model (DAG).

Graphical representations

In essence, even the most complex probabilistic models can be decomposed into the repeated application of *sum-* and *product-rule* (see Bishop chap. 1 [14]). However, this treatment can easily become tedious and confusing. To overcome these shortcomings, *probabilistic graphical models* have become very popular. Koller, Friedman have dedicated an extensive (landmark-) book on these models - [13]. Bishop lists three major advantages of graphical models:

1. *"They provide a simple way to visualize the structure of a probabilistic model and can be used to design and motivate new models."*
2. *"Insights into the model, including conditional independence properties, can be obtained by inspection of the graph."*
3. *"Complex computations, required to perform inference and learning in sophisticates models, can be expressed in terms of graphical manipulations, in which underlying mathematical expressions are carried along implicitly."*

quoted from [14].

When speaking of Bayesian networks, one usually refers to their graphical representation. As the name already suggests, the terminology arises from a network-like graphical representation. But of course, any joint-probability corresponds to a Bayesian network (not uniquely and it must result in an *acyclic* graph). A graph in general consits of *nodes/vertices* and *links/edges.* In case of probabilistic graphical models, the nodes correspond to random variables whereas the links represent probabilistic relationships between the variables. A Bayesian network must have *directed* edges and the graph must be acyclic, resulting in a **directed acyclic graph** (DAG). Another representation, called *Markov random fields* uses undirected edges and both representations can be converted into a so-called *factor graph*, which has advantages when it comes to inference via message passing (see [16]). To simplify the illustration of complex models, containing repetitions of some parts, the *plate notation* is commonly used.

To illustrate the graphical representation of a Bayesian network, consider the following example: Assume some manufacturing plant with a production process that requires a certain chemical reaction. The reaction takes place in a confined containment and will lead to an increased pressure within the containment - To prevent it from being damaged due to too high pressure, a sensor is constantly monitoring the current pressure. If the pressure rises above a certain threshold, an emergency shutdown will be initiated. However, the sensor could be damaged an return a false reading or the trigger could be faulty and initiate the shutdown at the wrong threshold. Therefore, all quantities shall be modeled as random variables: let the RV for the pressure within the containment be denoted by p, the RV for the sensor-reading by s and the RV for the (binary) output of the emergency shutdown trigger be t.

Based on these definitions, the joint-probability of the three involved variables is given by:

$$\begin{aligned} P(p,s,t) &= P(t|s,p)P(s|p)P(p) && (2.1a)\\ &= P(p|s,t)P(s|t)P(t) && (2.1b)\end{aligned}$$

Notice that the factorization of the joint-probability is not unique and eventhough Eq. (2.1a) is not the same as (2.1b) they both equal the same joint distribution. Since both factorizations are different, the resulting Bayesian networks will be different as well (same edges but different edge-directions). See Figure 2.4a for the graph corresponding to (2.1a) and Figure 2.4b for the BN constructed from (2.1b).

See Figure 2.4c for the Bayesian network that a human domain expert would have probably designed. One of the connections can be omitted, thus the graph has only two edges and corresponds to the following factorization of the joint-probability:

$$P(p,s,t) = P(t|s)P(s|p)P(p) \tag{2.2}$$

The first term on the right-hand side of the equation has been simplified since it is only conditioned on one variable. The factorization of Equation (2.1) is valid in general, whereas the factorization of Eq. (2.2) is only allowed if the (implicit) independence statement holds. In this case such an assumption is valid because the shutdown-trigger is based (entirely) on thresholding the sensor-reading - the trigger is not (directly) influenced by the actual pressure within the containment. A domain expert would probably know that there is no direct causal dependency between the pressure and the trigger and could use this knowledge to incorporate additional independence assumptions into the BN. In this case it would only lead to a small improvement, but one can easily see that in the context of large and complex models this could lead to significant simplifications compared to the naive factorization of a joint-distribution.

The model designed by the expert is *causal*, which makes it easier to understand but also simplifies the design-process. Notice though that all three models correspond to the same joint-probability - yet the model in Figure 2.4a does not show any causal relationships (the trigger-state does neither affect the sensor-reading nor the containment-pressure but). Furthermore, a model similar to 2.4c with inverted directions of the arrows would also be valid but it would

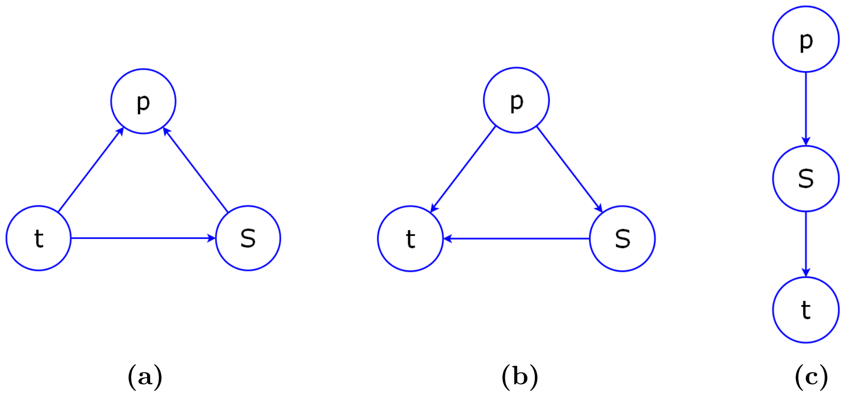


Figure 2.4: Bayesian network representation of a joint-probability - the pressure **p** in a containment is read via the sensor **s** and that reading is thresholded at the emergency-shutdown trigger **t**. (a): Graphical representation of Eq. (2.1a) - real-world causality is not reflected by the graph. (b): Representation of the same joint-probability but with a different ordering of the variables (see Eq. (2.1b)). (c): Byesian network designed by a human domain expert (Eq. (2.2)). Since the pressure will affect the trigger only via the sensor reading, the RVs **p** and **t** are (conditionally) independent (given **s**), thus there is no edge between these two nodes.

not reflect real-world causal relationships. This ambiguity is inherent to Bayesian networks as it is a direct result of the ambiguity of factorizations of a joint-distribution.

2.2.2 Inference and learning

The previous section has (briefly) introduced probabilistic (graphical) models, called Bayesian networks, which are capable of representing knowledge. But so far, the methods for "storing and inferring" knowledge, i.e. *learning and reasoning* have not been mentioned. As initially announced, one major advantage of probabilistic models is the separation of knowledge and reasoning - therefore many methods for both aspects have been found. The following introduction will only briefly mention the key-concepts behind most methods. Interested readers are referred to the standard-textbooks [14, 13].

Inference

Inference is the process of drawing conclusions from certain, known facts. However, in realistic scenarios these facts are usually governed by *uncertainty.* To capture this aspect, one usually resorts to probabilistic techniques, because the concept of pure logical reasoning collapses in the presence of uncertainty. The corresponding term is *probabilistic inference*, which refers to performing inference in probabilistic (graphical) models. In this context, the term *evidence* describes truly known facts or in probabilistic terms: observed values of certain random variables. Evidence plays an important role in the process of inference, since it forms the basis to draw conclusions from - however these conclusions are drawn in a probabilistic sense, according to a probabilistic model.

For Bayesian networks (or probabilistic graphical models in general) there are many approaches to perform inference. Some of them are exact (e.g. message passing), whereas many are approximative such as variational methods or sampling.

Message passing has been inspired by the topological structure of Bayesian networks - probabilities are sent as "messages" from one node to another and so on. Evidence can easily be incorporated in such messages. The "routing and processing" of messages is governed by a quite simple set of rules that, of course, has been mathematically motivated (marginalizations, sum- and product-rule, ...). Message passing in its essential form does not scale too well for "large" BNs. Improved versions of the approach exist and can be found under the names *belief propagation* or *expectation propagation* - see e.g. [14] or [17].

Another (very) popular approach is sampling - it is based on the assumption that any probability distribution can be characterized by drawing samples from it. In many practical problems it is very hard or even infeasible to provide an "analytic" expression for the underlying probability distributions. But with infinitely many examples drawn from the true distribution, the whole distribution could be described. In many cases, it is sufficient to draw a large number of samples and there are many methods if sampling directly from the true distributions is not possible. Sampling has been the topic of a lot of research not only in the context of probabilistic graphical models. However, most methods used in probabilistic inference are based on so called *Monte Carlo* methods. As this topic is vast and extensive, the reader is again referred to [18] or the standard-textbooks [14, 13] to gain a basic overview and as a starting-point for further literature research.

Learning the model (parameters)

Learning, in this case, refers to finding model-parameters such that the distribution resembled by the model is very similar to the true distribution that generated some (observation) data. In some cases, the model should be fitted as “tight” as possible to the given data, but in most cases this would be considered as some kind of overfitting-phenomenon. The interesting point is, that learning can be seen very similarly to inference, or at least as some kind of special-case of inference. By absorbing the model-parameters into a set of unobservable, latent random variables and treating the (observation- or training-) data as evidence - the problem of finding “good” values for the parameters can be solved by the process of inferring these values.

In principle, sampling or even message passing could be used to *learn* the model-parameters and actually there is a variety of algorithms based on these approaches. However, especially in terms of mixture models, expectation maximization (EM) has become very popular and it has widely been used (see [19] for an introduction on graphical models and learning with EM as well as Gibbs-sampling). EM is based on *estimating* the (log-) likelihood of the data given the current model-parameters (E-step) and in the subsequent M-step optimizing the model-parameters to maximize the (log-) likelihood. Both steps are repeated iteratively until some convergence criterion is met. The advantages of EM are its simplicity, solid theoretical foundations and a widespread use in many application domains - also, it might allow to derive closed-form (analytical) update equations which will lead to very fast implementations (compared to e.g. a sampling approach); however, this depends on the model and its complexity. EM was also used in the work of this thesis. An excellent introduction can be found in [14], which also provides nice examples for deriving the update equations with a (Gaussian) mixture model.

EM, or actually the *maximum-likelihood* approach, also has some shortcomings - regarding mixture models, the most severe drawback is its missing ability to infer the number of required mixture components (or “clusters”) from data. Variational methods are similar to EM and they overcome some of EM’s weaknesses - an introduction can also be found in [14] - a more extensive coverage is provided in [20]. The original proposal is in [21].

Learning the network-structure

Learning the model-parameters of a Bayesian network is only one aspect - it basically transforms (or compresses) the given data into a model that can be used to generate similar data, i.e. data with more or less the same distribution. But there is also another learning-aspect: learning the *topological* structure of the BN, i.e. the topology of the graph. This problem is significantly harder, since it can not be reduced to a parameter-fitting problem. In many cases, structural learning is avoided by simply using a fixed structure, designed or modelled by a human expert (engineer, domain-expert, etc.). This procedure has led to remarkable results, especially when consulting domain-experts, because they can usually provide knowledge on the causal relationships of the involved variables. However in many cases this is impossible.

To overcome this problem, methods for topological structure learning have been developed. Rather than having a "hand-crafted" model, the goal is to infer the model-structure as well as the parameters from the given data. This basically boils down to determining the connectivity structure (connection as well as direction) between the random variables. Since a connection indicates a conditional dependency, many approaches aim on determining whether such a dependency-hypothesis can be supported by the given data or not. However, the problem of determining such a dependency is ill-posed and in practice often hard to solve - especially when dealing with noisy data, it is hard to tell whether a (small) correlation between two random variables exists (resulting in a connection) or not.

As an alternative approach (sometimes also additional), there are methods that are based on a scoring function of some kind to evaluate the current model's quality. Starting with a model that has full connectivity, one can then (randomly) remove a connection and evaluate the quality of the new model. The problem is that this is computationally infeasible for large models (because if they were fully connected, the number of connections would simply be to large and the model-parameters have to be re-fitted on every trial). So, the approaches used in practice are often based on a combination of both approaches with (sometimes strong) heuristics.

There are also methods that involve the network-nodes as well. In some cases a model can significantly be simplified while maintaining or even improving its quality by introducing additional (latent) nodes. However, since these methods require learning of the connectivity structure as well as the number of nodes itself, they are computationally demanding and often too brittle to be used in a very general setting. In special cases, where the set of candidate models can be significantly limited, the methods can lead to nice results. See [22] for a quite sophisticated approach that also assumes structural regularities in the prior-distributions.

Topological structure learning can also be seen from a Bayesian point of view - in such a case one seeks to obtain a posterior for a set of prior-structures (candidate structures) and the given data. This problem can again be solved in many ways - a lot of solutions are based on a maximum likelihood approach, some of them using variations of expectation maximizations (structural EM like in [23, 24] or EM-MCMC for structure learning in [25]). For a more general treatment of the issue, see [13].

Topological structure learning is not necessarily equivalent to structural learning (extraction of structural invariants). This thesis, for instance, shows structural learning capabilities with a model that has a fixed topological structure. In this thesis, the topological learning aspects are not considered. However, there is a close relation and topological structure learning can also be of use when learning the structure of a task - e.g. if the structure involve some causality-relationships, these should be represented in the topological structure as well; or if the problem task is governed by a latent RV, this RV should appear in the network-structure as well.

Hierarchical Bayesian networks

Consider the following example: An experiment under certain conditions has led to a number of *noisy* measurements. Without any additionally knowledge, one would probably model these measurements with a Gaussian distribution (as a prior), having a mean- and variance-parameter. Now assume that the experiment is repeated under slightly different conditions -

leading to different mean- and variance-values. With several such trials, one could then again model these mean- and variance-values with a normal distribution (i.e. a prior on the prior-parameters). From a conceptual point of view, the latter Gaussian (prior) distribution lies on a more *abstract* level since it governs the parameters for all trials, whereas the initial Gaussian is responsible for "explaining" the values of the current trial. This results in a **hierarchical** model with two *layers of abstraction*.

A hierarchical Bayesian network simply consits of several layers of "priors", i.e. the parameters of the bottom-layer are governed by prior-distributions and the parameters of these again have their own priors, and so on. On a closer look, this is not different from a "regular" Bayesian network, since it corresponds to a joint distribution (including all the priors) and the corresponding graphical model. The term *hierarchical* rather refers to the structure of the model, having several layers of priors. Furthermore this allows (in many cases) a deeper interpretation of the individual layers, since they capture knowledge on several layers of abstraction. This is very interesting, since the upper levels of such a network correspond to more and more abstract or *general* knowledge. In machine learning, models that are capable of separating concrete and generalized knowledge are very desireable, because the general knowledge could potentially be "transferred" to similar tasks. Many traditional models do not allow such a separation or at least not that easily and thus provide a less useful basis when it comes to adapting to similar tasks (i.e. truly generalizing). For the example given above, it might actually be more reasonable to use a mixture-model as the more abstract prior since such a model is capable of clustering which could potentially lead to valuable (abstract) information on the problem-domain.

Notice that the hierarchical interpretation is not more than another point of view on Bayesian networks with certain topological features (not introducing many restrictions on the topology though). Yet it allows to gain deeper insights or design models that are more capable of resembling the "real-world" situation. Sometimes, the process of treating all model-parameters as random variables and introducing prior-distributions when required is referred to the *extension towards a full Bayesian model* - where a full model does no longer contain any deterministic parameter-values. Such models often show hierarchical features, but in many cases the focus is not on creating hierarchical Bayesian networks. *Variational methods* usually require a full Bayesian model and the extension towards such a model is often the first step in applying variational methods.

Another (big) advantage is that all the methods and techniques for Bayesian networks can still be applied to hierarchical models. Hierarchical Bayesian networks play a central role in this thesis, since the proposed model is hierarchical and the main interest lies on evaluating the capabilities of this model in terms of abstraction and generalization and further the exploitation of the abstract knowledge to rapidly adapt to novel but similar tasks. To gain a more intuitive insight on the aspects discussed in this section, consider the work of Kemp, Perfors and Tenenbaum [26] where they show a very nice and comprehensible example on what they call "learning overhypotheses" - which is just another, perhaps more suitable, term for extracting abstract knowledge.

3 Methods and models

This chapter will present the methods and (mathematical) models used in this thesis. It starts with a more precise definition of the problem task presented in the introductory chapter. In the following section, the basic hierarchical Bayesian model to extract the structure of the task will be introduced. In order to fit the model to observation-data (i.e. "learning"), the expectation maximization algorithm is used. The mathematical details of the EM-update equations can also be found in this section.

The basic model still has a few shortcomings (especially from a mathematical point of view). One of the main-issues is that the model-parameters do not have prior-distributions. In Section 3.3, the basic model will be extended (towards a full Bayesian model) by specifying priors for the model parameters. The reasons for this extension will be discussed in detail and the corresponding (mathematical) changes will be illustrated.

At the end of this chapter, the original problem task is refined into an *advanced* problem task. This advanced task is more complex from the perspective of planning (i.e. action-selection) It basically restricts actions to applying certain accelerations, which means that the agent can no longer set its velocity directly and has to plan a few steps ahead (e.g. if it needs to stay at a certain, close-by position but currently has a very high velocity).

To avoid any misconceptions: all implementations were done in Matlab - the agent as well as its environment are simulation only. The thesis does not use any real hardware to perform experiments.

3.1 Problem-task: distorted movement in discrete world

The problem-task described in 1.2 still needs some refinements in order to be suitable for the proposed models. For the discrete models, given in the following two sections, the task must be discretized. As a direct consequence, the 2D-plane where the agent moves is sectioned into a regular grid, where the agent can only move from grid-cell to grid-cell. Furthermore, the agent's velocities as well as the velocity-commands must only have discrete values. See Figure 3.1 for an illustration of the discretized problem-task.

The agent's position $\mathbf{p} = \langle x,y \rangle$ and velocity $\mathbf{v} = \langle v_x,v_y \rangle$ form the state of the agent $\mathbf{s} = \langle \mathbf{p},\mathbf{v} \rangle$. While this is perfectly valid, it will probably lead to computational problems with a discrete model. The computational demand usually grows exponentially with the number of different states. It is therefore desireable to reduce the state space as far as possible. The model proposed in the following section should learn the *state transition probability* $P(\mathbf{s}'|\mathbf{s},\mathbf{a})$ (where $\mathbf{a} = \langle v_x,v_y \rangle$ denotes an action and $\mathbf{s}'$ the subsequent state). In the end, this state transition probability will form the basis for the planning algorithm. But the state transition is *independent* of the global

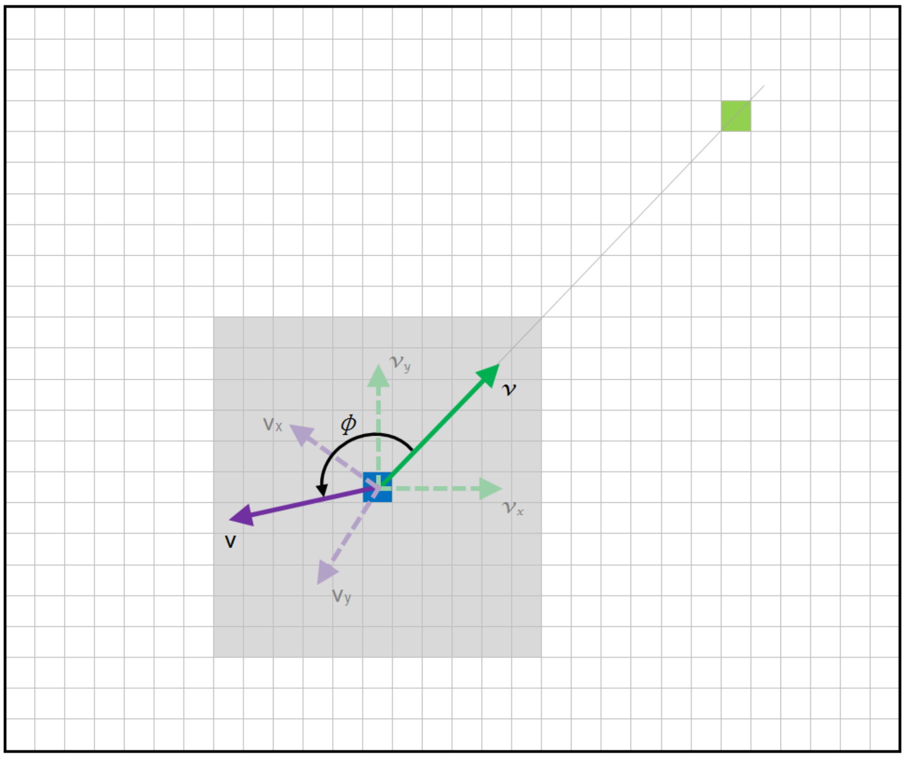


Figure 3.1: Discretized problem-task - the agent (in the blue shaded cell) tries to move towards the goal position (green shaded cell) in a “gridworld”. The corresponding discrete velocity-commands are shown in green (cursive captions). Due to the rotation distortion with angle ϕ, the actual velocity (purple, sans-serif captions) will point towards a completely different direction. Notice that velocities are discretized, i.e. they will always start and end at the center of a cell. Thus the actual velocity might have a different length (magnitude) than the corresponding command. The gray shaded cells show all possible subsequent positions - assuming velocity limits of $\pm$ 5.

position within the grid-world - thus only transitions relative to the current position need to be considered. See Figure 3.1 where the shaded area around the agent denotes all possible positions in the next step (considering all possible actions). Since the agent cannot reach a position outside the shaded area, all that needs to be considered are the shaded grid cells.

This results in a *relative transition model* with $\Delta x = x' - x = v_x$ and $\Delta y = y' - y = v_y$ - i.e. the cells that the agent can reach in its next step (relative to the current position) are identical to all possible velocities. The goal is then to learn: $P(\Delta x, \Delta y | v_x, v_y)$. This reduces our state-representation to: $\mathbf{s} = \Delta \mathbf{p} = \langle \Delta x, \Delta y \rangle$, i.e. the state of the agent in a relative transition model is independent of the global position of the agent!

Notice that the relative transition model can not be used to determine whether the agent is in the goal state or not. However it is trivial to expand the model such that it includes the global position; there is no need to incorporate this when *learning* the transition model.

The action space is formed by all possible actions, i.e. velocities. The state space for the relative transition model is formed by all possible (discrete) position-changes. Due to computational limitations it is inevitable to reduce the state and action spaces to “feasible” sizes by simply putting limits on the minimum/maximum velocity.

As stated in the original description of the problem task, the state transition model should not be deterministic. In order to incorporate some randomness, the actions are superimposed with a Gaussian noise, i.e. the actually selected action is a noisy version of the desired action. This results in the following transition model - where the noise as well as the overlaid rotation-distortion have been included:

$$P(\varDelta\mathbf{p}|\mathbf{a}) \propto \mathcal{N}([\underbrace{\varDelta x,\varDelta y}_{\varDelta\mathbf{p}}]^T|\mathbf{R}(\phi)[\underbrace{v_x,v_y}_{\mathbf{a}}]^T,\sigma^2 I), \tag{3.1}$$

where $\mathbf{R}(\phi)$ is the rotation matrix for angle ϕ.

This model can be used for simulation and *the generation of data-sets*. However, it is not related to the model used for learning the (relative) state transition **distribution**, described in the next section! Since the planning problem is trivial for the given setup, an advanced version of the problem will be presented in 3.4. In the advanced task, actions will be acceleration-commands, which makes it impossible to set the agent’s velocities directly.

3.2 Basic discrete model

The model presented in this section is a **hierarchical Bayesian model** that has been proposed by Maass, Neumann, Rückert (IGI) and has mainly been inspired by the work of Kemp, Perfors, Tenenbaum [26]. It is able to extract structural knowledge from the given data in the form of prior-distributions.

3.2.1 Multinomial representation of the data

For each discrete action $\mathbf{a}_n = \langle v_x,v_y\rangle$ and each possible position change $\varDelta\mathbf{p}_l$ the transition function can be represented by a multinomial distribution: $P(\varDelta\mathbf{p}_l|\mathbf{a}_n,\boldsymbol{\theta}) = \theta_{ln}$. By definition $\sum_l \theta_{ln} = 1$, i.e. for each action a_n, there is a normalized parameter vector $\boldsymbol{\theta}_n$.

Assume N datasets $D_j = \{\langle\varDelta\mathbf{p}_{jl},\mathbf{a}_{jn}\rangle\}$ with $1 \leq j \leq N$ have been acquired, where the angle of the rotation-distortion is kept constant for an entire set but may vary among different sets. Each dataset corresponds to a single episode of the agent, i.e. the recorded data of a number of subsequent steps performed in the grid-world environment. The likelihood of D_j is then given by

$$P(D_j|\boldsymbol{\theta}) = \prod_{n=1}^{|A|}\prod_{l=1}^{|\varDelta P|} \theta_{ln}^{m_{ln}}, \tag{3.2}$$

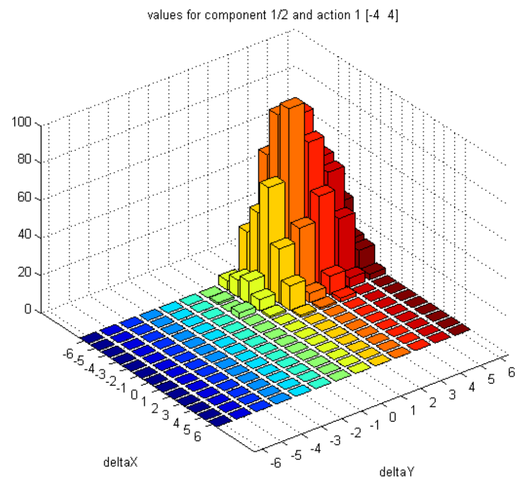
where m_{ln} is the number of examples within the dataset where action $\mathbf{a}_n$ was taken and resulted in a position change $\varDelta\mathbf{p}_l$. $|A|$ is the number of discrete actions and $|\varDelta P|$ is the number of (valid, discrete) position changes - in this case (with a relative transition model) it is identical to the number of possible velocities.

Figure 3.2 shows exemplary plots for $\mathbf{m}_l$. The values have been acquired from two datasets - each one has been generated using a different rotation-angle. In the plots, the elements of $\mathbf{m}_l$ (the vector has $|\varDelta P|$ entries), have been mapped onto a two-dimensional grid, that represents the x- and y-components of $\varDelta\mathbf{p}$. The figure also illustrates the non-deterministic nature of the commands - repeatedly using the same command with the same rotation-distortion, will produce a regime of actually taken velocities. The variance within this regime is governed by the σ-parameter of (3.1).

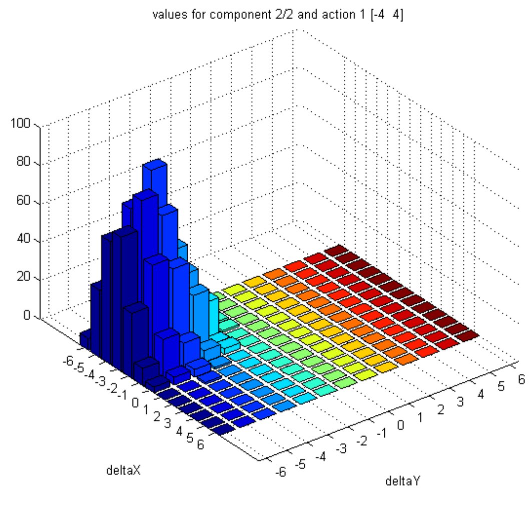
3.2.2 Dirichlet-mixture as prior distribution

In order to be able to incorporate structural knowledge into the model, it desirable to introduce general or prior knowledge about the parameters of the multinomial distributions. In a Bayesian manner, this is done by using *prior distributions*. The normalized product with the likelihood of the data will then form a *posterior distribution*, which incorporates a prior belief as well as the actual observations. In a hierarchical Bayesian model the abstract, generalized knowledge is modeled with prior *distributions*.

Similar to the model proposed in [26], the model proposed by Maass, Neumann, Rückert will use the Dirichlet distribution as a prior. The Dirichlet distribution is a *conjugate prior* for the multinomial distributions, which means that the posterior distribution will have the form of a Dirichlet distribution as well. Furthermore, the hyperparameters of the Dirichlet have a nice interpretation as an *"effective number of observations"* (see [14] Chapter 2.2.1).



(a)



(b)

Figure 3.2: Parameters of multinomial distribution - values of $\mathbf{m}_l$ for the action $\mathbf{a} = [-4{,}4]$ from two datasets, generated with different rotation angles - (a): 6° and (b): 99°. The plots show how many times the action has led to a particular change in position $\varDelta\mathbf{p}_i = [\varDelta x, \varDelta y]$. Notice that the same action will result in similarly shaped distributions and how the rotation angle can be seen with the naked eye. The data was generated using quite a high variance for drawing the actual position change (see (3.1)).

A single Dirichlet distribution will most probably not be *expressive* enough to capture the structure within the data that should be learned. Instead it would simply average over the different rotation-distortions. The key idea, proposed by Mass, Neumann, Rückert is to use a **mixture of Dirichlet distributions** for the prior distribution. Then, the prior for $\boldsymbol{\theta}$ has the following notation:

$$P(\boldsymbol{\theta}|\boldsymbol{\alpha}_{1:K},\mathbf{c}) = \sum_{k=1}^{K} \mathrm{Dir}(\boldsymbol{\theta}|\boldsymbol{\alpha}_k)P(k|\mathbf{c}), \tag{3.3}$$

with

$$\mathrm{Dir}(\boldsymbol{\mu}|\boldsymbol{\alpha}) = \frac{1}{W(\boldsymbol{\alpha})}\prod_{k=1}^{K}\mu_k^{\alpha_k-1}, \tag{3.4}$$

and the normalization/weighting function:

$$W(\boldsymbol{\alpha}) := \frac{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)}{\Gamma(\alpha_0)}, \tag{3.5}$$

and

$$\alpha_0 = \sum_{k=1}^{K}\alpha_k, \tag{3.6}$$

where K is the number of mixture components, $\Gamma(x)$ denotes the gamma-function and $P(k|\mathbf{c})$ acts as a weight of mixture component k:

$$P(k|\mathbf{c}) = c_k \tag{3.7}$$

All elements of $\mathbf{c}$ must sum up to one ($\sum_{k=1}^{K} c_k = 1$).

A more schematic view of the model can be seen in Figure 3.3. At the bottom you can see the N datasets, where the rotation distortion within a dataset is constant. The observations of each dataset are modeled with a multinomial distribution to have a probabilistic representation of the data. On the next, more abstract level, a mixture of Dirichlet distributions acts as a prior for the multinomials. Notice the general assumption that the mixture has (a lot) less components than the number of datasets, i.e. $K < N$. This is not crucial for the model itself, but besides leading to overfitting effects, too many mixture components might also lead to problems like degenerated or singular cases when fitting the model to data using expectation maximization (EM - see 3.2.3). At the top-most level there is an additional variable $\mathbf{c}$ which is simply a weight for the Dirichlet mixture components and indicates the *importance* of the individual component for the given datasets.

It is very common to introduce a *latent* or *hidden* variable to mixture distributions - see Bishop chap. 9.2 [14] for a detailed example with a mixture of Gaussians. The latent variable $\mathbf{z}$ is a binary vector with only one element being equal to 1 - also known as a "1-of-K representation". It ensures that a datapoint is assigned to a single cluster-component. In combination with the EM algorithm the use of hidden variables will lead to significant simplifications (the latent variable is explicit and each mixture component can be optimized independently - see Bishop for a discussion). For the model used in this thesis, the only difference from this general concept

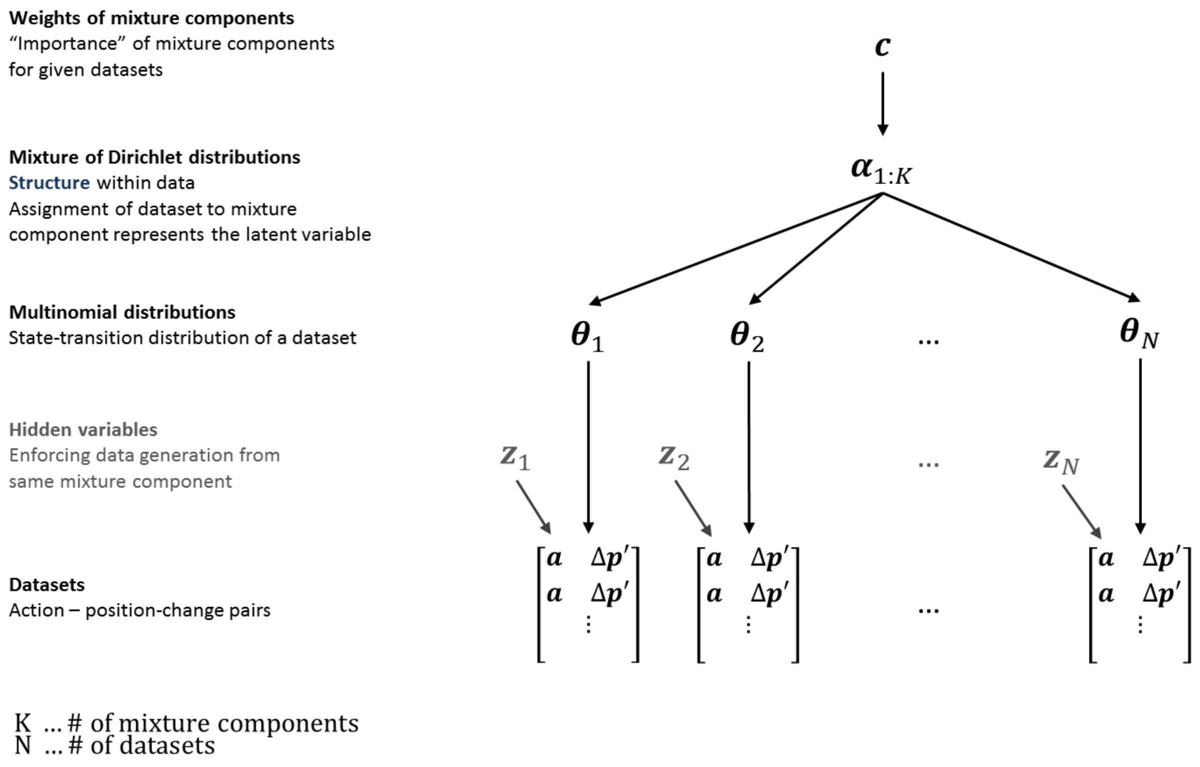

Weights of mixture components
"Importance" of mixture components for given datasets
Mixture of Dirichlet distributions
Structure within data
Assignment of dataset to mixture component represents the latent variable
Multinomial distributions
State-transition distribution of a dataset
Hidden variables
Enforcing data generation from same mixture component
Datasets
Action – position-change pairs
c
$\alpha_{1:K}$
θ_1 θ_2 ... θ_N
z_1 z_2 ... z_N
a $\Delta p'$
K ... # of mixture components
N ... # of datasets

Figure 3.3: Basic discrete model - the observations in the datasets are modeled using multinomial distributions. Abstract knowledge is incorporated via the prior distribution - a mixture of Dirichlet distributions. At the "top level" there is an additional (prior) weight. To implicitly define that all observations within a dataset have been generated from the same mixture component, the hidden variable z is introduced.

is, that the latent variable is used per dataset and not per datapoint. This is valid because of the assumption that all observations within a dataset have the same rotation-distortion, i.e. they can (and actually should) be explained by the same mixture component. Or from another point of view: the latent variable determines from which mixture component of the prior the current task was generated, thus directly corresponding to the structure of the task.

Using the latent variable and the previous definitions, the joint distribution of the data D, the hidden variable $\mathbf{z}$ and the parameters of the mixture distribution $\boldsymbol{\alpha}, \mathbf{c}$ is given by:

$$P(D_{1:N}, \mathbf{z}_{1:N} | \boldsymbol{\alpha}_{1:K}, \mathbf{c}) = \prod_{j=1}^{N} \prod_{k=1}^{K} \underbrace{\left(P(k|\mathbf{c}) \int_{\boldsymbol{\theta}} P(D_j|\boldsymbol{\theta}) P(\boldsymbol{\theta}|\boldsymbol{\alpha}_k) d\boldsymbol{\theta} \right)^{z_{jk}}}_{P(z_{jk}, D_j | \boldsymbol{\alpha}_k, \mathbf{c})} \tag{3.8}$$

Because the latent variable is a binary vector, which has only a single element that is not equal to zero, the sum in (3.3) can be written as a product.

3.2.3 Fitting the model to data with EM

In order to extract the structure of the N datasets, the model-parameters $\boldsymbol{\alpha}, \mathbf{c}$ need to be fitted to the given data. There is a number of different approaches - each having its own strengths and weaknesses. In this case, the *expectation maximization algorithm* (EM) was chosen; on one hand it is not too complex, yet powerful and the mathematical demands seem feasible. On the other hand the algorithm is well known, commonly used and has solid theoretical foundations (see [27] for a brief tutorial, [14] for a more detailed discussion and [21] for the original proposal of the algorithm). Especially in conjunction with mixture models, EM has become very popular to estimate the (hidden/latent) model parameters from observation-data (see also [28]). It is not within the scope of this thesis to evaluate the qualities of EM for the given problem, nor to compare it with different algorithms.

The EM-algorithm consists of two subsequent steps that are iteratively executed until some convergence criterion is met. In the so called *E-step* the **expectation** of the likelihood (of the data, given the model-parameters) is computed, with the current estimates of the model-parameters. In the following *M-step*, the model-parameters are optimized to **maximize** the expected (log-) likelihood.

The following section shows the analytical derivation of equations to compute the required quantities. Unfortunately, no closed-form solution for the maximization in the M-step could be found - however it was possible to derive the corresponding gradient, thus being able to *increase* the expected log-likelihood by using gradient-ascent (this is also known as *generalized* EM).

E-step

The goal of the E-step is to compute the expected value of the likelihood of the latent variable $\mathbf{z}$ given the data and assuming fixed model-parameters. In conjunction with a *mixture*-model, this quantity is referred to as the so-called *responsibility.* The responsibility basically gives a measure of how likely it is that a dataset D_j belongs to mixture component k (or has been generated from it). During the computation of these quantities, the latent variable $\mathbf{z}$ can be exploited - i.e. the responsibility of each mixture component can be computed separately (up to a normalization along all mixture components).

The responsibility $q_j(k)$ of dataset j belonging to mixture component k is given by:

$$q_j(k) = P(z_{jk} = 1|D_j,\boldsymbol{\alpha}_k,\mathbf{c}) = \frac{P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c})}{P(D_j|\boldsymbol{\alpha}_k,\mathbf{c})} = \frac{P(k|\mathbf{c}) \int_{\boldsymbol{\theta}} P(D_j|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{\alpha}_k)d\boldsymbol{\theta}}{\sum_k P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c})} \tag{3.9}$$

The following only considers the multinomial parameters for a single dataset j therefore the index j will be omitted - i.e. $\boldsymbol{\theta} = \boldsymbol{\theta}_j$ as well as $\mathbf{m}_{j,n} = \mathbf{m}_n$. Also, only a single mixture component k is considered - with the hyperparameters $\boldsymbol{\alpha}_n = \boldsymbol{\alpha}_{k,n}$ for each action $\mathbf{a}_n$ (and $\boldsymbol{\alpha}_{nl} = \boldsymbol{\alpha}_{k,nl}$); again the index k has been omitted.

The denominator of Eq. (3.9) is just a normalization factor - a closer look on the computation of the numerator (using the previous definitions) yields:

$$\begin{aligned}
P(k|\mathbf{c})\int_{\boldsymbol{\theta}} P(D|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{\alpha})d\boldsymbol{\theta} &= P(k|\mathbf{c})\left(\int_{\boldsymbol{\theta}}\prod_{n=1}^{|A|}\prod_{l=1}^{|\Delta P|}\theta_{ln}^{m_{ln}}\frac{1}{W(\boldsymbol{\alpha}_n)}\prod_{l=1}^{|\Delta P|}\theta_{ln}^{\alpha_{ln}-1}d\boldsymbol{\theta}\right)\\
&= P(k|\mathbf{c})\prod_{n=1}^{|A|}\left(\frac{1}{W(\boldsymbol{\alpha}_n)}\int_{\boldsymbol{\theta}}\underbrace{\prod_{l=1}^{|\Delta P|}\theta_{ln}^{m_{ln}+\alpha_{ln}-1}}_{unnormalizedDirichlet}d\boldsymbol{\theta}\right)\\
&= c_k\prod_{n=1}^{|A|}\left(\frac{W(\mathbf{m}_n+\boldsymbol{\alpha}_n)}{W(\boldsymbol{\alpha}_n)}\underbrace{\int_{\boldsymbol{\theta}}\mathrm{Dir}(\boldsymbol{\theta}|\mathbf{m}_n+\boldsymbol{\alpha}_n)d\boldsymbol{\theta}}_{=1}\right)
\end{aligned} \tag{3.10}$$

The multinomial distribution with the Dirichlet as its conjugate prior, results in a Dirichlet distribution. However it is unnormalized and after introducing the correct weighting-term, integrating over the (now normalized) Dirichlet simply yields 1.

The final result for the responsibility $q_j(k)$ and thus for the E-step is:

$$q_j(k) = \frac{P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c})}{\underbrace{\sum_k P(z_{jk},D_j|\boldsymbol{\alpha}_{1:K},\mathbf{c})}_{normalization}} = \frac{c_k\prod_{n=1}^{|A|}\frac{W(\mathbf{m}_{j,n}+\boldsymbol{\alpha}_{k,n})}{W(\boldsymbol{\alpha}_{k,n})}}{\sum_k P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c})} \tag{3.11}$$

M-step

In the M-step, the expected value of the (complete-data) log-likelihood (of the latent variable) is to be maximized. The following approach (as well as a more detailed derivation and discussion) can be found in [14], chap. 9.

$$\begin{aligned}
\mathbb{E}_{\mathbf{z}}\left[\ln P(\mathbf{z},D|\boldsymbol{\alpha},\mathbf{c})\right] &= \ln\left(\prod_{j=1}^{N}\prod_{k=1}^{K}P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c})^{P(z_{jk}=1|D_j,\boldsymbol{\alpha}_k,\mathbf{c})}\right)\\
&= \sum_{j=1}^{N}\sum_{k=1}^{K}q_j(k)\ln P(z_{jk},D_j|\boldsymbol{\alpha}_k)\\
&= \sum_{k=1}^{K}\sum_{j=1}^{N}q_j(k)\left(\ln P(k|\mathbf{c})+\sum_{n=1}^{|A|}\ln\left(\frac{W(\boldsymbol{\alpha}_{k,n}+\mathbf{m}_{j,n})}{W(\boldsymbol{\alpha}_{k,n})}\right)\right)
\end{aligned} \tag{3.12}$$

Notice that each mixture component can be optimized separately (again due to the latent variable z). Also notice that $\mathbf{c}$ and $\boldsymbol{\alpha}$ can be optimized independently.

To find an update equation for $\mathbf{c}$, optimize the following (for all K mixture components):

$$\sum_{k=1}^{K}\sum_{j=1}^{N} q_j(k) \ln P(k|\mathbf{c}) \tag{3.13}$$

with respect to c_k

Remember that $P(k|\mathbf{c}) = c_k$. As a side-constraint $\mathbf{c}$ must sum up to one ($\sum_k c_k = 1$) - therefore a La-Grange multiplier is used. This derivation is quite common and can also be found in Bishop's example (see [14] for an analytical derivation). The result is simple and intuitive and has the following form:

$$c_{k,new} = \frac{\sum_j q_j(k)}{\sum_\kappa \sum_j q_j(\kappa)} \tag{3.14}$$

The weight of a mixture component depends on "how responsible" the component is with respect to *all* datasets ("How many datasets are explained by this particular mixture component?"); the denominator is just a normalization-term.

To find the parameters of the Dirichlet prior, optimize the following with respect to $\boldsymbol{\alpha}_k$ for all K mixture components (note that it is valid to consider only a single mixture component, since they can be optimized separately):

$$\sum_{j=1}^{N} q_j(k) \sum_{n=1}^{|A|} \ln W(\boldsymbol{\alpha}_{k,n} + \mathbf{m}_{j,n}) - \ln W(\boldsymbol{\alpha}_{k,n}) \tag{3.15}$$

Unfortunately there is no closed-form solution for this maximization. However, it is possible to compute the gradient (with respect to $\boldsymbol{\alpha}_k$) and use gradient-ascent to increase the log-likelihood. This is also known as generalized expectation maximization (GEM - see also: [27]), where the M-step is not guaranteed to maximize the log-likelihood but simply increase it. The derivation of the following result can be found in the appendix A.1:

$$\nabla \alpha_{k,nl} = \sum_{j=1}^{N} q_j(k) \left(\psi(\alpha_{k,nl} + m_{j,nl}) - \psi(\alpha_{k,nl}) + \psi(\alpha_{0k,n}) - \psi(\alpha_{0k,n} + R_{j,n})\right), \tag{3.16}$$

with

$$R_{j,n} := \sum_{l=1}^{|\Delta P|} m_{j,nl}, \tag{3.17}$$

i.e. the number of times that action n was taken in dataset j and $\psi(x)$ being the *digamma* function (the derivative of the log-gamma function $\Gamma_{\ln}$). $\alpha_{0k,n}$ is the sum over all α-entries for a single action and mixture component:

$$\alpha_{0k,n} := \sum_{l=1}^{|\Delta P|} \alpha_{k,nl} \tag{3.18}$$

The update equation for $\boldsymbol{\alpha}_k$ then has the following form:

$$\boldsymbol{\alpha}_{k,new} = \boldsymbol{\alpha}_k + \lambda_{\mathrm{LR}} \cdot \nabla \boldsymbol{\alpha}_k \tag{3.19}$$

where λ_{LR} denotes a certain *learningrate.*

Additional details on the implementation

In practice, the EM-algorithm has a few common pitfalls - especially in conjunction with mixture-models. One of the main issues is that the number of mixture components is not clear - too few components will probably lead to a model that is not expressive enough to capture fine-grained structural aspects. On the other hand, too many mixture components will cause unnecessary computational demands as well as overfitting effects. One task of this thesis is therefore to investigate the effects of different numbers of mixture components - as it turns out (see the results in Chapter 4 for more detail) the model is quite robust for a range of mixture component counts and the exact number of components is not too problematic as long as it lies within the range of about 7 up to 20 components.

Another problem, inherent to EM and mixture models, is that EM will only be able to find parameter settings for local maxima (of the likelihood) and thus, the results will strongly depend on the initial values. Furthermore, bad choices of initial values might lead to degenerate cases, where a single mixture-component collapses onto a single dataset (i.e. the component is only *responsible* for a single dataset). In most cases, this is undesired and EM in its basic form is usually not able to recover from such cases. As a simple heuristic rule, the following convention was used during EM-iteration: When a single mixture component becomes responsible for a number of datasets that is less than a third of the datasets of the the mixture component with the most datasets associated, **delete** the degenerate mixture component and **copy** the one with the most datasets with some minor noise on the parameters.

The EM-update equations will lead to normalized values for the mixture-component weight vector $\mathbf{c}$. The hyper-parameters of the Dirichlet distributions, $\boldsymbol{\alpha}$, are driven towards the values of the corresponding multinomial parameters (of all dataset where the mixture component has a significant responsibility). Most of these values are zero whereas the others might have double- or triple-digit values (see also figure 3.2). Both, too small values (close to zero) and too large values (above 80) are problematic, since they lead to numerical issues. In the basic model, the values must therefore be limited to a certain range (e.g. $(0.001, 25)$).

3.3 Full Bayesian model with hyper-priors

From a conceptual as well as a mathematical point of view, it is desired to have prior distributions for the parameters of the basic model ($\boldsymbol{\alpha}$). In a Bayesian sense, this corresponds to having a "full" Bayesian model - however one could also view these priors on the hyper-parameters (hyper-priors) as another level of abstraction, eventhough it is hard to give them an intuitive interpretation in terms of capturing abstract knowledge.

The introduction of such hyper-priors also promises to overcome some of the problems of the basic model, as mentioned at the end of the previous section. First of all, limiting of $\boldsymbol{\alpha}$-values should no longer be necessary - at least at the upper bound (the lower bound is close to zero and at least in some cases, a hard limit might still be needed). Another problem of the basic model is that it might produce prior-distributions ($\boldsymbol{\alpha}$) that show a degenerated structure (instead of a nice bells-shape, like the multinomials in figure 3.2, the learned $\boldsymbol{\alpha}$'s might have outliers, very large values or almost all non-zero values at their limits). The following section describes an extension to the basic model that reduces these problems while maintaining a conceptual/mathematical elegance (in a Bayesian sense).

3.3.1 Hyper-priors

To overcome the problem of having to limit the individual $\boldsymbol{\alpha}$-values, a **mixture of Beta distributions** is used. The mixture consists of two components - one for the values at the lower limit, close to zero, and another one for the values that have significant nonzero values. As these nonzero values might have quite a different range among the different Dirichlet distributions, it is reasonable to use normalized $\boldsymbol{\alpha}$-values, i.e. normalize the values with the sum over all values for a single action of a mixture component (see Eq. (3.21)). Since the lower-bound of $\boldsymbol{\alpha}$-values is close to zero, the Beta-prior might not have a strong enough limiting effect on these values and it is possible that a single Beta distribution for the significant nonzero values is sufficient. The following model will therefore use a general notation for the Beta-mixture and the simulations will show, whether a single distribution or a mixture is needed.

To further enforce that the learned $\boldsymbol{\alpha}$'s for each action are similarly shaped, another prior distribution is introduced. It acts on the $\boldsymbol{\alpha}_0$'s, i.e. the sum over all values of a particular action. The prior for this sum is a **Gaussian distribution** with a certain mean and variance. If a Dirichlet component has a degenerate "shape", where either the sum or the variance over all values is quite different from all other Dirichlet components, its likelihood will be quite low and its parameters will be driven to be more similar to the parameters of the other distributions (in the M-step).

Figure 3.4 shows a schematic sketch of the extended Bayesian model - the parameters of the (new) Beta-mixture prior are denoted by $\boldsymbol{\lambda}$ and the corresponding mixture weight is $\mathbf{x}$; the parameters of the Gaussian prior (mean and variance) are subsumed in the variable $\boldsymbol{\gamma}$.

As mentioned before, the hyper-prior distributions are hard to intuitively interpret - furthermore it is hard to find reasonable values for the parameters of these hyper-priors (parameters of the Beta distributions as well as mean and variance of the Gaussian). These parameters will therefore also be fitted to the given data (using EM) during the learning-stage.

With the Beta-mixture prior on the normalized $\boldsymbol{\alpha}_k$ and the Gaussian prior on $\boldsymbol{\alpha}_{0k}$ we get:

$$\boldsymbol{\alpha}_k \longrightarrow P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa}) = \prod_{n=1}^{|A|}\prod_{l=1}^{|\Delta P|} \underbrace{\left(\sum_{b=1}^{B} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b)P(b|\mathbf{x})\right)}_{Beta-mixture} \underbrace{\mathcal{N}(\alpha_{0k,n}|\boldsymbol{\gamma})}_{Gaussian}, \qquad (3.20)$$

with the hyper-prior parameter vector $\boldsymbol{\kappa} = \langle\boldsymbol{\lambda}, \mathbf{x}, \boldsymbol{\gamma}\rangle$.

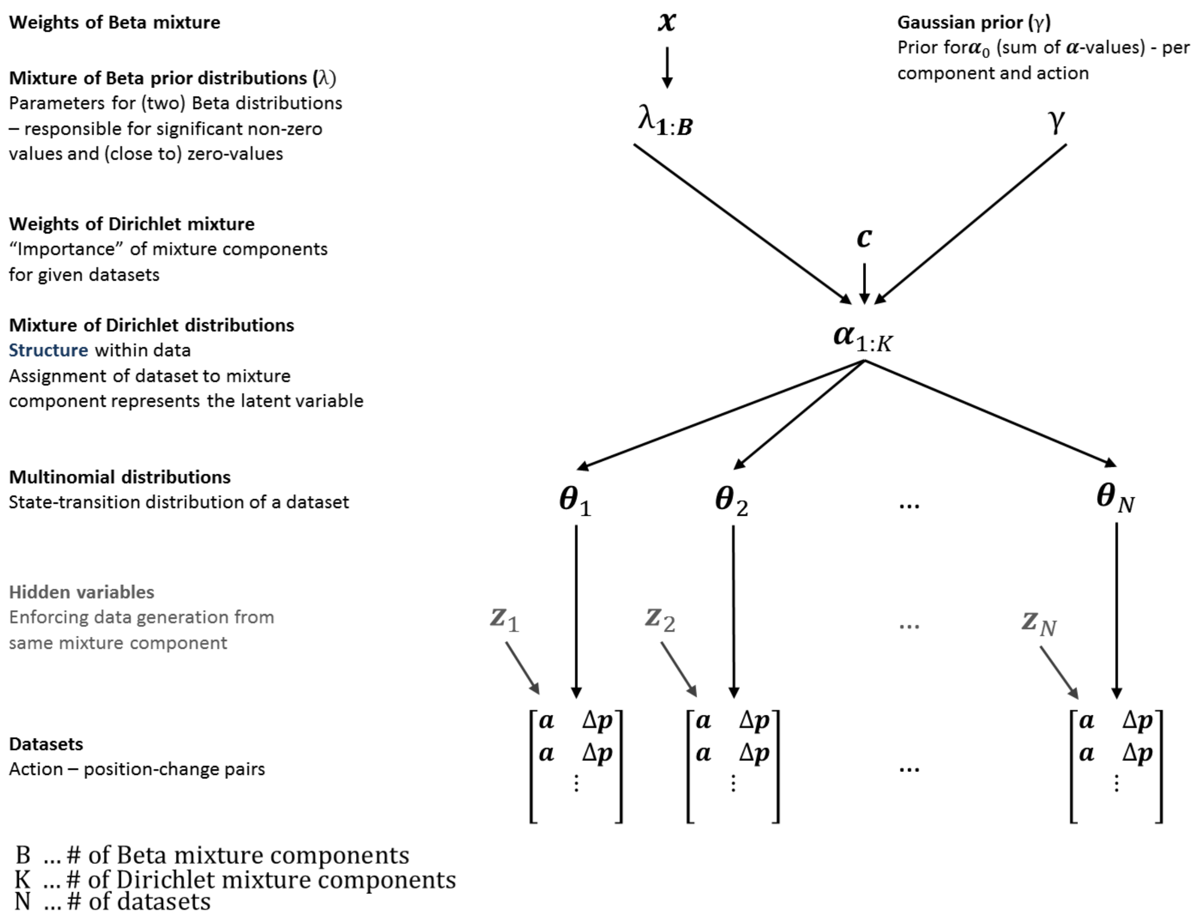

Weights of Beta mixture
Mixture of Beta prior distributions (λ)
Parameters for (two) Beta distributions – responsible for significant non-zero values and (close to) zero-values
Weights of Dirichlet mixture
"Importance" of mixture components for given datasets
Mixture of Dirichlet distributions
Structure within data
Assignment of dataset to mixture component represents the latent variable
Multinomial distributions
State-transition distribution of a dataset
Hidden variables
Enforcing data generation from same mixture component
Datasets
Action – position-change pairs
Gaussian prior (γ)
Prior for $\boldsymbol{\alpha}_0$ (sum of $\boldsymbol{\alpha}$-values) - per component and action
$\boldsymbol{x}$
$\lambda_{1:B}$
γ
$\boldsymbol{c}$
$\boldsymbol{\alpha}_{1:K}$
$\boldsymbol{\theta}_1$
$\boldsymbol{\theta}_2$
$\boldsymbol{\theta}_N$
$\boldsymbol{z}_1$
$\boldsymbol{z}_2$
$\boldsymbol{z}_N$
$\boldsymbol{a}$ $\Delta\boldsymbol{p}$
B ... # of Beta mixture components
K ... # of Dirichlet mixture components
N ... # of datasets

Figure 3.4: Extended hierarchical Bayesian model - the parameters of the Dirichlet-mixture prior now have additional (hyper-)prior-distributions: A mixture of Beta distributions as well as a Gaussian prior for the α_0-values.

B is the number of Beta-mixture components - since it is not clear whether a single Beta distribution is sufficient or a mixture of two Betas is needed. $\mathbf{x}$ is the weight-vector for the Beta-mixture (compare this to $\mathbf{c}$ which is the weight-vector for the Dirichlet-mixture). $\boldsymbol{\lambda}$ represents the parameters of the Beta-distributions and $\boldsymbol{\gamma}$ denotes the parameters of the Gaussian prior (mean and variance).

The argument of the Beta-distribution is given as:

$$\mu_{k,nl} = \frac{\alpha_{k,nl}}{\alpha_{0k,n}} \tag{3.21}$$

where $\alpha_{0k,n}$ is defined in Eq. (3.18). Thus, $\boldsymbol{\mu}$ represents *normalized* $\boldsymbol{\alpha}$-values (normalized per action and mixture component).

3.3.2 EM for the extended model

This section shows the changes in the EM-equations compared to the basic model with respect to $\boldsymbol{\alpha}$. The updates for the hyper-prior parameter vector $\boldsymbol{\kappa}$ can be found in Sec. 3.3.3. Since the hyper-priors extend the basic model, large parts of the EM-equations remain the same and in most cases additional (additive) terms have to be introduced.

E-step

The E-step basically consists of computing the responsibility of the mixture components for a dataset D_j. The prior-probability $P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})$ introduces slight changes to the compared to the E-Step of the basic model (see Sec. 3.2.3 or Eq. (3.11)).

The responsibility $q_j(k)$ of dataset j belonging to mixture component k of the extended model is given by:

$$q_j(k) = P(z_{jk} = 1|D_j,\boldsymbol{\alpha}_k,\mathbf{c},\boldsymbol{\kappa}) = \frac{P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c},\boldsymbol{\kappa})}{P(D_j|\boldsymbol{\alpha}_k,\mathbf{c},\boldsymbol{\kappa})} = \frac{P(k|\mathbf{c})P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})\int_{\boldsymbol{\theta}} P(D_j|\boldsymbol{\theta})P(\boldsymbol{\theta}|\boldsymbol{\alpha}_k)d\boldsymbol{\theta}}{\sum_k P(z_{jk},D_j|\boldsymbol{\alpha}_k,\mathbf{c},\boldsymbol{\kappa})} \tag{3.22}$$

M-step

In the M-step, the parameters of the model are optimized to maximize the expected log-likelihood (in the case of generalized EM the log-likelihood is increased but not maximized). The following, considers the optimization of the $\boldsymbol{\alpha}$-values, the optimization of the hyper-prior parameters ($\boldsymbol{\kappa}$) is shown in the next section.

The (complete-data) log-likelihood now has the following form:

$$\mathbb{E}_{\mathbf{z}}\left[\ln P(\mathbf{z},D|\boldsymbol{\alpha},\mathbf{c},\boldsymbol{\kappa})\right] = \sum_{k=1}^{K} \underbrace{\sum_{j=1}^{N} q_j(k)\ln P(z_{jk},D_j|\boldsymbol{\alpha}_{1:K})}_{\textit{same term as in basic model}} + \ln P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa}) \tag{3.23}$$

Compare this equation to the log-likelihood of the basic model ((3.12)). In the extended model, every mixture component has an additional hyper-prior term $P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})$, given by Equation (3.20). Notice that the hyper-prior term is independent of the dataset D_j and thus can be pulled out of the inner sum.

The mixture components can still be optimized independently and due to the ln the products of Eq. (3.20) factorize nicely, rendering the α-parameters for each action independent of the ones for all other actions. The detailed derivations for the optimization of $P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})$ with respect to $\boldsymbol{\alpha}_k$ are somewhat tedious and can be found in the appendix (A.3).

Notice that there is again no closed-form solution for the $\boldsymbol{\alpha}$-update but a closed-form solution for the corresponding gradient, which allows the application of gradient ascent for optimizations (see Eq. (3.19)).

3.3.3 Optimization of hyper-prior parameters

As mentioned previously, it is hard to intuitively find *good* parameter-settings for the hyper-priors. Therefore, these parameters shall be optimized, using the given datasets, as well. The basis for this optimization is again the (complete-data) log-likelihood function (Eq. (3.23)). Unfortunately, it is not possible to find a closed-form solution for the update equations of the Beta-parameters. It is not even possible to provide a closed-form solution for the corresponding gradient, which is required for optimization. However, there is still a possibility to apply gradient ascent - by using *numerical methods* to estimate the gradient.

Given a function $f(x)$, depending on the variable x, the gradient with respect to x can be numerically estimated by using a small deviation $\varDelta$ around an operating point x_0:

$$f'(x_0) \approx \frac{f(x_0 + \varDelta) - f(x_0 - \varDelta)}{2\varDelta} \tag{3.24}$$

which corresponds to a linearization of the function in a small neighborhood around the operating point and then computing the "slope" of the function. This is a fairly crude method and also the exact value for the deviation $\varDelta$ is not clear. Additionally there is an issue with gradient ascent (in general) - the optimization will converge toward a local maximum and the final solution is therefore (strongly) dependent on the initial values.

The optimization of the parameter-settings for the Gaussian prior can be done by using the closed form solution. The result is well known but it can also be intuitively motivated: since the goal of the hyper-prior is to enforce a certain similarity among the different Dirichlet-parameters, it is reasonable to set the mean-value of the Gaussian to the mean over all Dirichlet distributions (taking each action individually), i.e. the mean over all $\alpha_{0k,n}$. The variance of the Gaussian is set in the same way. Without further proof, it is easy to see that the likelihood of the $\boldsymbol{\alpha}_0$'s will increase the closer the Gaussian parameters are to the mean and variance of the $\boldsymbol{\alpha}_0$'s (assuming that most $\boldsymbol{\alpha}_{k,n}$ will have a similar shape). Furthermore, degenerated Dirichlet priors, will lead to worse values of the log likelihood function.

In both cases, the parameters to be optimized need to be bounded by upper and lower limiting-values. Otherwise, gradient-ascent might drive the parameter-values towards very small or very large values which become numerically problematic for the simulations, but might also lead to unfortunate results for the $\boldsymbol{\alpha}$-values.

3.4 Advanced problem-task

The basic problem task, described in 3.1 uses velocity-commands as actions. This makes the corresponding planning-problem trivial, since the agent must only plan the immediate next step. Therefore, the planning algorithm only has to search for a locally optimal policy for each state. In order to make the planning-problem non-trivial, an advanced task setup where the actions are acceleration-commands, is now introduced. This requires *"planning a few steps ahead"*, since the agent has to decelerate when it gets into the vicinity of the goal-position.

Now, it is no longer valid to always choose a locally optimal action (the one that gets the agent closest to the goal position, given its current position) but the planning-algorithm has to search for a globally optimal policy.

3.4.1 Modifications

Figure 3.5 shows an overview of the modified task setup. Instead of directly setting its velocity, the agent must now apply an acceleration to the current velocity. The rotation-distortion acts between the desired acceleration (-command) and the actually taken acceleration. The hierarchical Bayesian model should again extract the rotation-distortion as the *structure* of the task. Velocities and accelerations as well as the positions of the agent on the gridworld are discrete.

Actions are now defined as $\mathbf{a} = \langle a_x, a_y \rangle$ and they will result in a velocity-change $\mathbf{v}' - \mathbf{v} = \Delta\mathbf{v}$. In every time-step dt, the agent at position $\mathbf{p} = \langle x, y \rangle$ will perform a transition to the position $\mathbf{p}'$ according to the following probabilistic **transition-model**:

$$\begin{aligned} P(\mathbf{p}',\mathbf{v}'|\mathbf{p},\mathbf{v},\mathbf{a}) &= P(\mathbf{p}'|\mathbf{v}',\mathbf{v},\mathbf{p})P(\mathbf{v}'|\mathbf{v},\mathbf{a}) \\ &\propto \mathcal{N}\left(\mathbf{p}'|\mathbf{p} + \mathbf{v}dt + \frac{\mathbf{v}' - \mathbf{v}}{2}dt, \sigma_p^2\right)\mathcal{N}\left([v' - v]^T|\mathbf{R}(\phi)\mathbf{a}^T, \sigma^2\right) \end{aligned} \tag{3.25}$$

The latter part of the equation is very similar to Equation (3.1) (the transition model of the basic task). It is very important to notice, that the rotation-distortion only affects that part of the model; the first Gaussian-term that models the position-transition for a given position, velocity and velocity-difference has **no** rotation-distortion. This allows to ignore this part of the model when learning the structure of the problem-task - all that needs to be considered with the hierarchical Bayesian model is the latter part of Equation (3.25). Notice that the planning-algorithm has to take the full model into account. Essentially, this means that for the learning problem he exact same model as in the basic task can be used. In case of the new problem setup the model is simply given a different *interpretation*. When planning, the posterior, composed of the data and the learned priors can be used for $P(\mathbf{v}'|\mathbf{v},\mathbf{a})$ (see (3.25)).

In general the timestep dt should be fairly small, in order to have a realistic model. However, this would lead to problems due to discretization and limited sizes of state- and action-space. For instance, if the agent was currently at velocity (0,0) and an acceleration of (1,0) would be applied for a timestep of $0.1s$, then the corresponding change in position would probably not suffice to end up in a different grid-cell. Due to discretization, the recently applied acceleration would simply vanish and the agent could never reach a non-zero velocity. One way to solve this issue was the use of quite large accelerations, probably up to values of 20 or 30, but even with a coarse discretization, this would significantly increase the size of our action-space. This is problematic from a computational point of view - especially when using value-iteration for planning (see 3.4.2). The other solution is to simply use large timesteps - in the simulations of this thesis a timestep of $1s$ will be used, which leads to more unrealistic models. However, for the purpose of evaluating the qualities of an HBM for structural learning this has no significant impact.

Since the grid-world has a (quite) limited size, the boundary conditions have to be well defined. This problem task uses the following **convention**: all movements that would put the agent

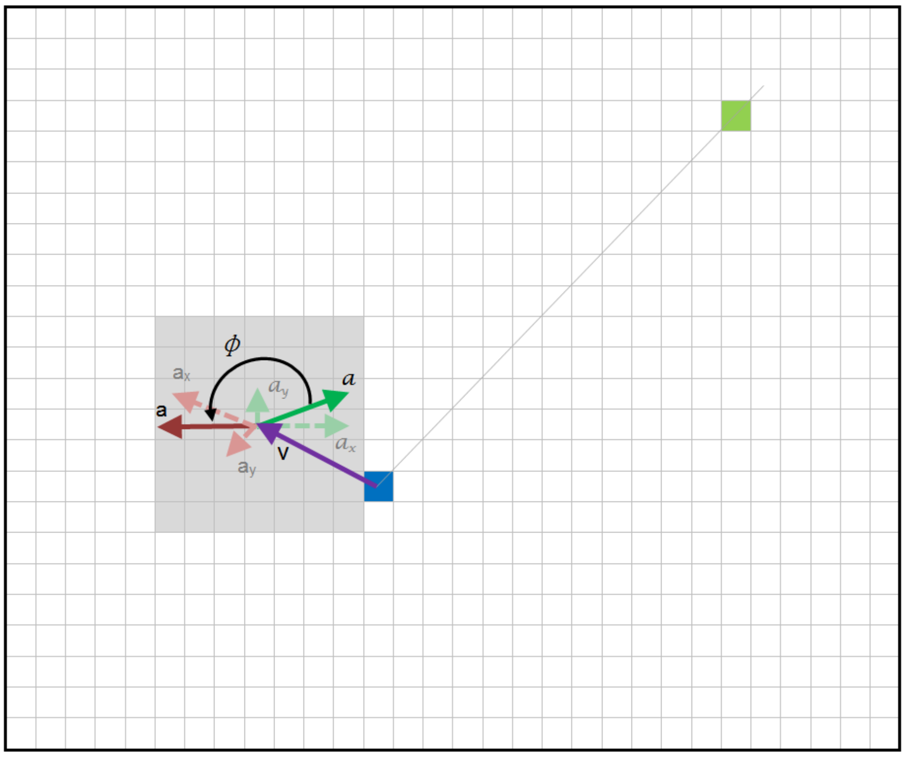


Figure 3.5: Problem-task with accelerations as actions - The agent invokes a certain acceleration-command (green, cursive captions) to change its current velocity. Due to the rotation-distortion with angle ϕ, the actually applied acceleration (red, sans-serif captions) points towards a different direction and because of the discretization may also have a different magnitude than the original command. The gray shaded cells show all possible accelerations, assuming an acceleration-limit of $\pm$ 3.

outside the boundaries of the grid-world, will be limited to the last position within the grid-world; i.e. our grid-world has "walls" along the boundaries. However, the agent will simply keep its velocity when bumping into a wall. In a similar way, the velocity of the agent is limited - accelerations that would result in velocities that exceed the limits, will set the velocity to that limit-value.

3.4.2 Planning with value-iteration

In order to solve the problem of finding a (globally) optimal policy that will allow the agent to choose actions such that is reaches the goal-state, some kind of planning-algorithm is necessary. Since the transition model is probabilistic (and noisy), the algorithm must be able to cope with such models. One basic, yet powerful such algorithm is *value-iteration*. In its original form it is quite simple; however it does require a full model of the environment (in this case the full state-transition model). In case of discrete models, this can quickly turn into computationally infeasible problems - simply due to the computational explosion that comes with large state- and action-spaces. The problem is well known and there are several improved versions or different algorithms that do not require a (full) model of the environment. However, value-iteration is sufficient for the demands that are within the scope of this thesis and has the advantage of simplicity, thus easy implementation. You can find a very brief overview of value-iteration in A.2.

The only restrictions introduced by using value-iteration regard the size of the state- and action-spaces. For the advanced problem-task this leads to the restriction to fairly small gridworlds with only a handful of (discrete) velocities and acceleration-commands.

Besides the full state-transition model, a *reward function* is necessary for planning - especially when using value-iteration. It is defined as $R(\mathbf{s},\mathbf{a})$, which requires the *design* of a function that returns a reward-value for every state-action pair. In the given problem-task the state $\mathbf{s}$ is a composed of the agent's position and velocity ($\mathbf{s} = \langle \mathbf{p},\mathbf{v} \rangle$) and the actions $\mathbf{a}$ are acceleration commands.

For this problem-task, the agent's position should be as close as possible to the goal position, while at the same time the agent should only accelerate/decelerate when necessary, which could be interpreted as a constraint to operate economically. Let the goal position be $\mathbf{p}_G$, then the reward function for this task shall be defined as:

$$Q(\mathbf{s},\mathbf{a}) = Q(\langle \mathbf{p},\mathbf{v} \rangle,\mathbf{a}) := -|\mathbf{p} - \mathbf{p}_G|^2 - 0.1|\mathbf{a}|^2 \tag{3.26}$$

where $|\cdot|$ in this case denotes the L1-norm $||\cdot||_1$. Note that this choice is arbitrary - one could also choose the L2-norm on any other applicable norm and the reward does not necessarily need to increase quadratic with the distance. In this case the L1-norm seemed to better reflect the circumstance that the agent can only move cell-wise.

Notice that this is rather a design than a unique definition because there are many different ways to define the reward-function such that the agent will navigate towards the goal position and try to stay there. The optimal policy, in order to reach that goal, will strongly depend on the design of the reward function. Furthermore, the reward function will also have an impact on the convergence-properties of value-iteration.

There is one special-case, where this reward function will not lead to the desired behavior of the agent: if the goal-position is at the edge of the gridworld (i.e. next to a "wall") and the agent reaches that cell with a velocity that points towards the wall, there is no need to decelerate, since the agent can not leave the grid-world and will stay at the goal-position. Furthermore, there is a penalty on every action, depending on the magnitude of the acceleration. The agent would therefore receive the highest (expected) reward by applying the "smallest" possible acceleration ([0,0]). To avoid this special-case, the reward for being at the goal position with **zero-velocity** has been increased by plus one.

4 Results

The models described in the previous chapter are put to test using the problem-tasks defined there as well, to obtain some insight on the qualities of the models in terms of *structural learning*. The ability for *fast learning* by exploiting the learned structure of the task, captured by the priors of the model, is of special interest. The first section of this chapter provides more abstract results on this issue, whereas the second section presents more intuitive insights on the gridworld-navigation task. If not mentioned otherwise, the full Bayesian model (the extended model) was used to produce the results.

In both cases, the findings were obtained in a two-stage process. In the initial learning stage, a number of generated datasets was used to provide a basis for extracting the structure of the task. Each dataset was created in a separate episode with a different angle for the rotation-distortion and consits of state-action pairs, that denote into which state a certain action has lead. The action-selection during data-generation is purely random. In order to extract the task structure and *learn abstract, transferable knowledge*, the hierarchical model is fitted to the generated data, using the expectation maximization algorithm.

In the subsequent evaluation-stage, one or more new episodes with novel angles (that have not been presented before) are generated. These new episodes are then used to perform the analysis of certain qualities of the model. In this stage, the model is no longer fitted to the novel data, but the learned priors are used in conjunction with the new data to obtain a *posterior distribution* that should rapidly (i.e. after a few steps) be very close to the true transition-distribution of the new episode, because it has the same underlying structure. In terms of the agent's movement this means, that it should only take a few steps until the agent is able to compensate the rotation-distortion and navigate towards the goal position.

The final section of this chapter provides some insight on the process of optimizing the hyper-prior parameters during the learning stage.

4.1 Basic task

For the basic task setup (as described in 3.1), the goal is to evaluate the capabilities of the model in terms of extracting the task structure. Since the planning-problem for this task is trivial, the analysis will be restricted to more abstract issues. Section 4.2 provides more intuitive results showing the agent's movement behavior under various settings.

4.1.1 KL-divergence analysis

To evaluate the ability for fast learning, a measure for the difference between the true distribution of an episode and the posterior distribution, composed of the learned priors and the episode's data, is needed. As a standard tool, the *Kullback-Leibler divergence* is used - for discrete distributions it is defined as:

$$\mathrm{KL}(q\|p) = -\sum_{i} q(i) \ln \left(\frac{q(i)}{p(i)} \right) \tag{4.1}$$

Since the KL-divergence is not symmetric, i.e. $\mathrm{KL}(q\|p) \neq \mathrm{KL}(p\|q)$ the arithmetic average of both versions has been used.

The posterior distribution is the product of the prior-distribution with the likelihood of the current dataset (modeled as a multinomial distribution). Since the observation-data is modeled with a multinomial distribution and the prior is the conjugate Dirichlet distribtuion, the postrerior is a Dirichlet distribution as well. Let the parameters of the new data be $\mathbf{m}_j$ and the parameters of the Dirichlet prior be $\boldsymbol{\alpha}_k$. The resulting posterior for *a specific action* $\mathbf{a}_n$ would then be a Dirichlet distribution with the sum of the multinomial- and the prior-parameters: $\mathrm{Dir}(\mathbf{m}_{j,n} + \boldsymbol{\alpha}_{k,n})$.

In the case of a Dirichlet-*mixture* prior, all mixture components need to be incorporated into the prior. This is done by using the corresponding responsibilities $q_j(k)$. In order to use the posterior for planning, the expected value of the posterior with respect to a particular position-change $\varDelta\mathbf{p}_l$ needs to be computed. The expectation of a Dirichlet distribution $\mathrm{Dir}(\boldsymbol{\alpha})$ with respect to the i-th parameter is given by $\frac{\alpha_i}{\alpha_0}$, where α_0 is the sum over all elements of $\boldsymbol{\alpha}$.

The posterior-probability of ending up with a position-change $\varDelta\mathbf{p}_l$ when taking action $\mathbf{a}_n$, given the priors $\boldsymbol{\alpha}$ and the hyper-priors $\boldsymbol{\kappa}$ as well as the observations of a single dataset D_j is given by:

$$P(\varDelta\mathbf{p}_l|\mathbf{a}_n,D_j,\boldsymbol{\alpha}_{1:K},\boldsymbol{\kappa}) = \sum_{k=1}^{K} q_j(k)\mathbb{E}_{\varDelta\mathbf{p}_l}\left\{\mathrm{Dir}(\boldsymbol{\alpha}_{k,n} + \mathbf{m}_{j,n})\right\} = \sum_{k=1}^{K} q_j(k)\frac{\alpha_{k,nl} + m_{j,nl}}{\alpha_{0k,n} + R_{j,n}}, \tag{4.2}$$

where $\alpha_{0k,n}$ is defined in (3.18) and $R_{j,n}$ is defined in (3.17). To show the evolution of the KL-divergence, a new value is computed after each step of an episode.

You can see the results Figure 4.1 and Figure 4.2, which show the evolution of the KL-divergence over the first steps of a novel episode using different numbers of mixture components. The true transition distribution of the new episode has been estimated by simply using a large number of steps/data-points. To have more reliable estimates of the KL-evolution, the plots have been obtained by averaging over 10 episodes. Each episode was created with a random angle for the rotation distortion and all models (with different mixture component counts) were tested against the same set of episodes.

Figure 4.1 shows the evolution of the Kullback-Leibler divergence over the first 1500 steps of a new episode as well as the first 50 steps, as they are of special interest for fast learning. For the initial learning stage, 60 datasets with random angles were used. In this first experiment the action set was chosen to be particularly small - there are only two possible actions: $(-4, 4)$

and $(4, -4)$. This should simplify the overall learning problem and it significantly reduces computational efforts. The novel episodes, used for KL-comparison, were created with the same action-set and 10000 steps for estimating the true transition-distributions for each episode were used. The results were finally averaged over 10 trials.

The results in Figure 4.1 show that the KL-divergence decreases rapidly, when using a model with a high number of mixture components. This suggests that a model with a low number of mixture components will start to average over a broad range of angles for the rotation-distortion, thus is not capable of correctly capturing the structure of the task. However, compared to the plots for data-only (without using the priors of the model) even a single Dirichlet prior will lead to a better KL-divergence. On a closer look, this is not very surprising since the data-only model simply lacks observations. Especially in the early phase of an episode, the data-only distribution still contains zero-probability values for many transitions that will actually occur (i.e. have a nonzero value in the true distribution). The models using a prior have compressed all transitions that they experienced in the learning stage into the prior. Thus a model even with a single mixture component has already nonzero transition probabilities for many transitions that occur in the true distribution. On the other hand, it would also have nonzero probabilities for transitions that will have zero-values in the true distributions, because the single mixture component model has averaged over many different rotation angles. But the latter does not have such a significant impact on the KL-divergence as the absence of transitions for the data-only model.

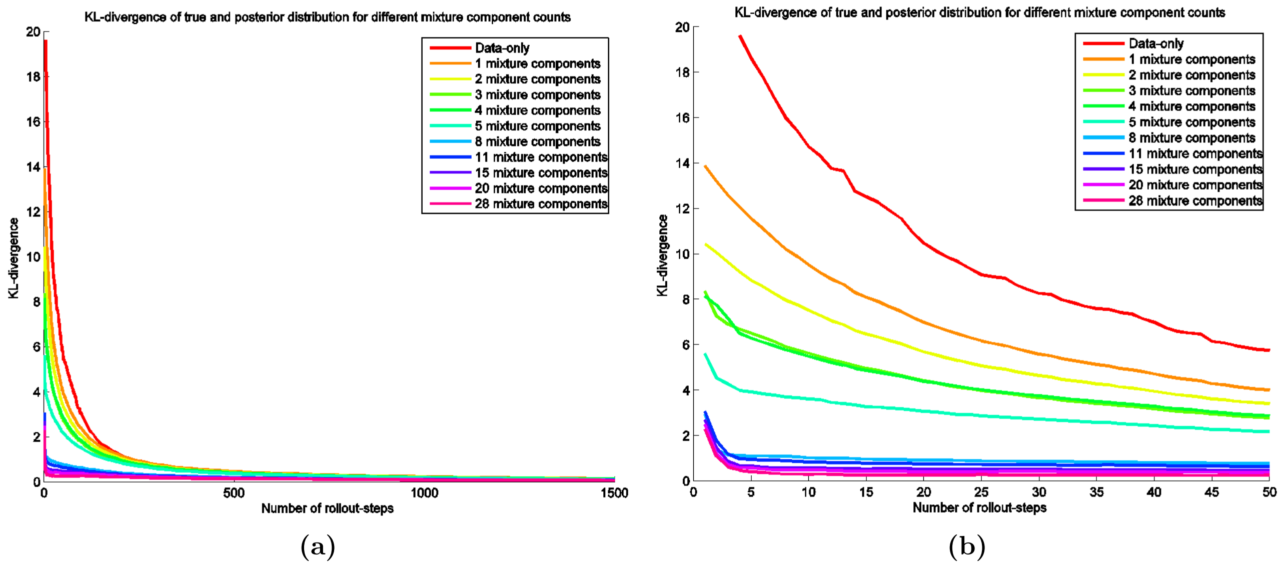


Figure 4.1: KL-divergence analysis for small action-set - Evolution of the KL-divergence for different mixture-component counts. (a) shows the evolution for the first 1500 steps - (b) shows the first 50 steps in detail. If the number of mixture components is large enough, the KL-divergence decreases very rapidly. Both plots also show the KL-divergence for a model without any priors (data-only) which requires many steps to to reach small values.

Figure 4.2 shows the results of the same analysis, but this time with a large action-set (any action within the limits of $(-4, 4)$). Compared to Figure 4.1 the overall convergence towards the true distribution takes more steps (notice that the plot now shows the first 10000 steps). Also the behavior within the first 50 episodes looks less smooth. Both effects are results of the larger action-set - it now takes a lot more steps to actually execute the same action multiple times (compared to having only two actions) and there are some actions with a certain ambiguity - e.g. the action $(0, 0)$ will have the same transition-distribution under all rotation-angles. But the qualitative results remain the same: the hierarchical model significantly decreases the number of steps required for a "good" posterior distribution. The effect becomes more prominent for a larger the number of mixture components - however, a certain baseline of deviation from the true distribution seems to remain. This is also a result of the large action-set: the learned priors do not have the exact same distribution as the true distribution of the novel data; which leads to a nonzero KL-divergence. With a very large number of steps, the posterior should be mainly influenced by the data and the effects of the prior should more or less vanish - using the described action-set, this takes (a lot) more than 10000 steps.

For the learning stage, 50 datasets with random angles and for estimating the true distribution episodes with 250000 steps were used. As in the previous figure, the results were averaged over 10 trials. The large number of steps for the learning-sets is required to ensure that the transition distributions of the learning-episodes are close to the corresponding true distributions. Otherwise the model would simply learn from the wrong distributions.

The key result of this section is the demonstration of *fast learning* when using the learned priors of the hierarchical model. If the model has enough mixture components and a sufficient ammount of training-data and thus is able to capture the structure of the task, only very few

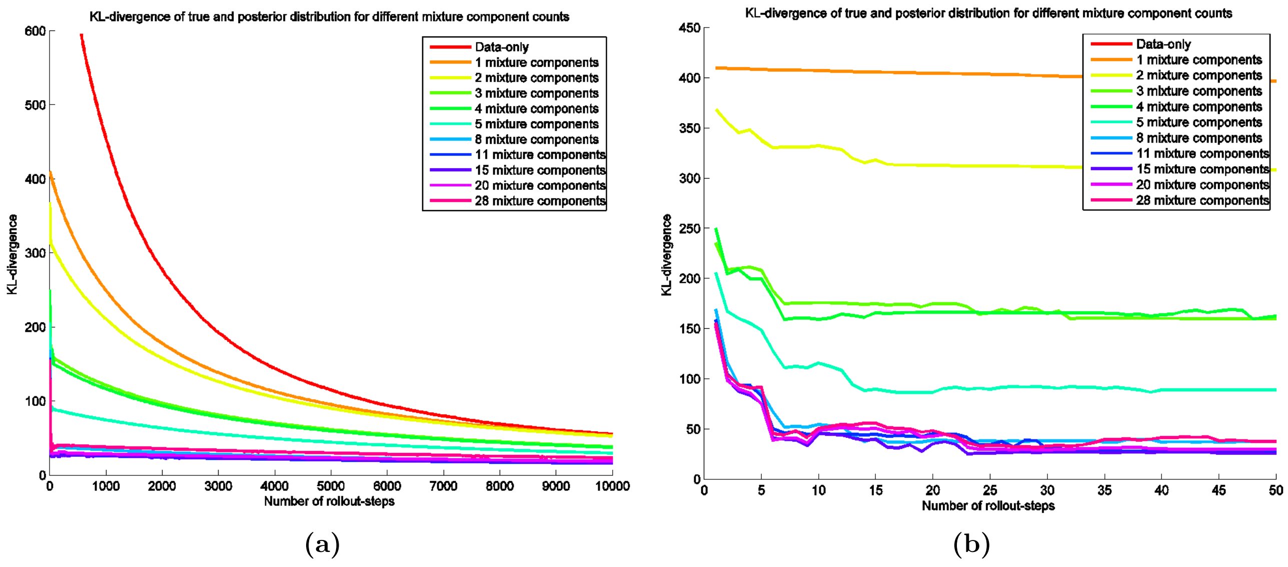


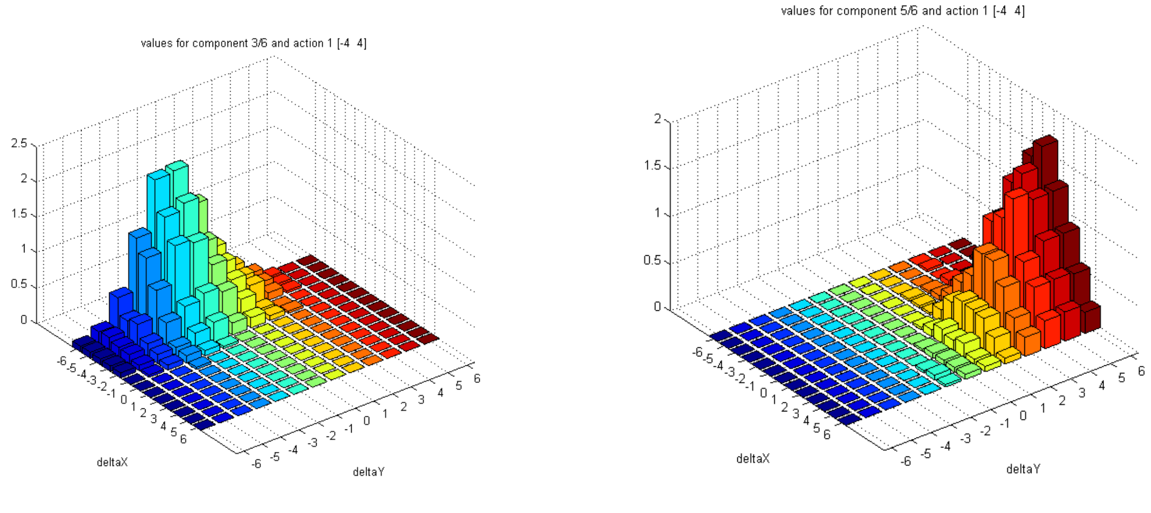
Figure 4.2: KL-divergence analysis for full action-set - Evolution of the KL-divergence for different mixture-component counts using an action set with all possible values between -4 and 4 (on each axis). (a) shows the evolution for the first 10000 steps - (b) shows the first 50 steps in detail.

steps are necessary to obtain a posterior distribution that is very close to the true distribution of the data. Compared with using the observation-data only (without the learned priors) the number of steps for having a “useful” posterior, e.g. for planning, is **significantly** reduced.

4.1.2 Learned priors

To gain some more insight on how the structure is extracted by the hierarchical model, a closer look on the learned prior-distributions might help. Actually, the plots in Figure 4.3, Figure 4.4 and Figure 4.6 show the hyper-parameters $\boldsymbol{\alpha}$ of the Dirichlet mixture components. These parameters can also be interpreted as virtual observations (i.e. observations that represent the prior belief). For a single dataset, these parameters should have the same (distribution-) shape as the corresponding multinomial-parameters of the dataset. For several datasets, the shape should resemble the “average” over all multinomials which are explained by that particular mixture component.

Figure 4.3 shows the learned hyper-parameters of two different Dirichlet mixture components. The shown parameters correspond to a single action $(-4, 4)$. In this case 60 episodes with random angles were used for the learning stage. The total number of mixture components was chosen to be six - which leads to a *clustering* of the angles of rotation-distortions into six clusters. The components have been mapped onto a 2D-grid that matches the x- and y-components of the corresponding velocity. Compare these results with the parameters of the multinomial distributions (of the data) - see 3.2.1.



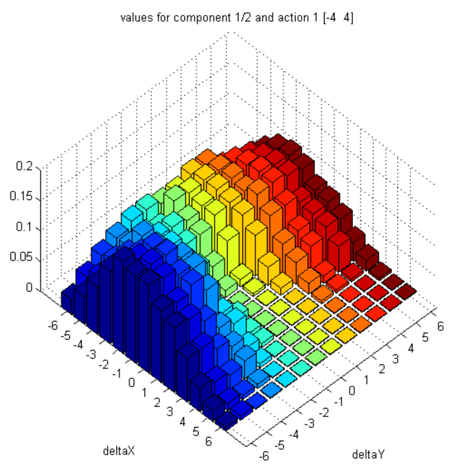
(a) (b)

Figure 4.3: Learned Dirichlet hyper-parameters - 6 mixture components - (a) shows the results for a mixture component that has its cluster-center at about 60°. (b) shows a mixture component with a center at 287°. The clusters are quite well shaped and do not seem to show large averaging effects - the results of the KL-divergence analysis (4.1.1) show that 6 mixture components are probably not enough and that there are still some slight averaging effects.

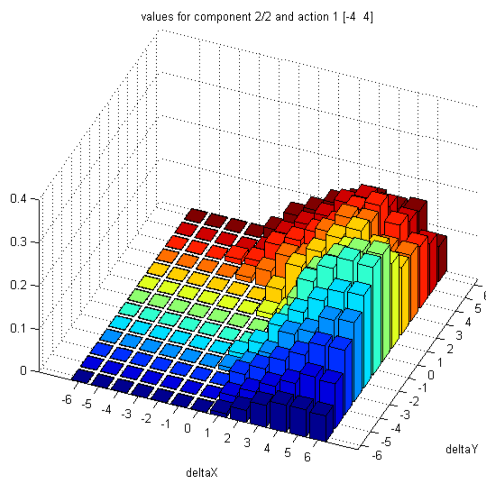
In Figure 4.4 you can see the results for a model that uses only **two** mixture components - in this case, significant averaging effects over a subregime of the structure are visible - also notice the magnitudes of the values on the z-axis, which are now much smaller (this could be interpreted as a higher degree of uncertainty). Of course this is an extreme example to show the effects, because two mixture components are obviously not enough to capture the structure of rotation-distortions over the whole range of angles between zero and 360 degrees.

Figre 4.5 illustrates the evolution of the responsibilities of the mixture components. The responsibility is a measure of how likely the current dataset has been generated from a particular mixture component. 4.5a shows the evoultion for the model with six mixture components (see 4.3 for illustrations of the learned Dirichlet hyper-parameters $\boldsymbol{\alpha}$). After the first step, the responsibilities are not very distinct and the wrong mixture component actually gets the highest responsibility. But after only four steps, the correct mixture component gets (almost) full responsibility. In 4.5b the responsibilities are correct after the first step already - however there are only two mixture components and as long as the angle of the new dataset does not lie directly between the two clusters, assigning the correct responsibility is not too brittle.

This section, as well as the previous section, emphasizes the dependency on the number of mixture components - if there are too few components, the model is not able to correctly capture the structure of the task or in other words: the extracted structure is not representative for the given data. Too many components, on the other hand, will lead to larger computational efforts as well as (slight) overfitting effects. The choice of the number of mixture components is not obvious, however the results suggest that eight to 20 components are a reasonable choice.

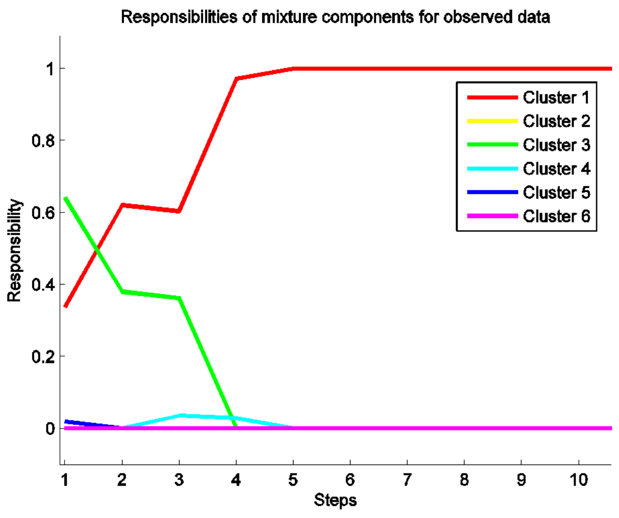


(a)

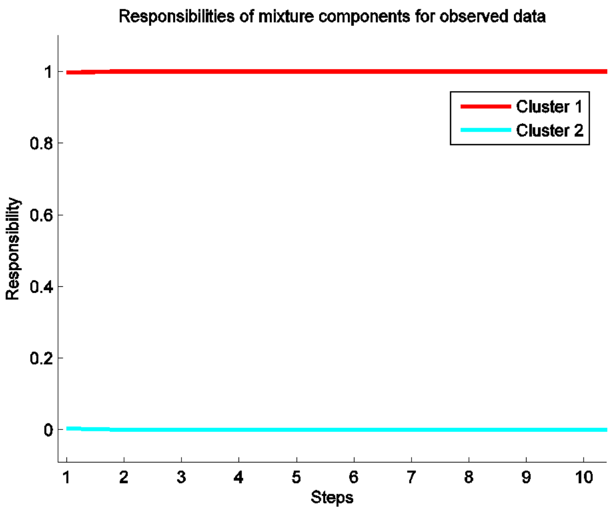


(b)

Figure 4.4: Learned Dirichlet hyper-parameters - 2 mixture components. Strong averaging effects are visible - the learned priors show a quite different distribution-shape, compared to the multinomial parameters of the observations.



(a)



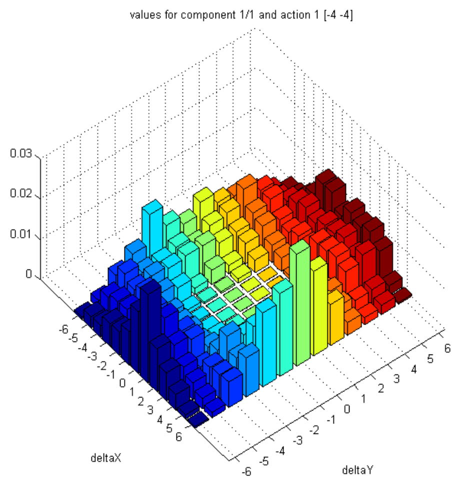
(b)

Figure 4.5: Evolution of responsibilities - (a): Evolution for the model with six mixture components. After the fourth step mixture component 1 (cluster 1) gets more or less the full responsibility, which is correct in this case. (b): Responsibilities for the model with two mixture components. Cluster 1 gets (more or less) full responsibility after the first step already. Despite the averaging effects caused by too few mixture components, the responsibilities are correct.

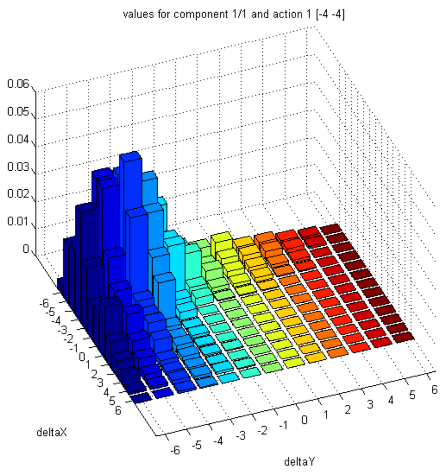
4.1.3 Posterior distribution

Figure 4.6 illustrates the evolution of the posterior (transition) distribution over the first steps of a new episode. More precisely the figure shows the hyper-parameters of the posterior distribution for a single action, projected onto a 2D grid that represents the corresponding velocity-changes. The initial posterior, before performing a single step, is almost uniform (the central velocities have a very small probability since the action is $(-4, 4)$). Initially, the responsibilities of the mixture components for the novel dataset are unclear (i.e. the rotation-cluster is unknown) and therefore all mixture components are more or less equally probable which results in an almost uniform posterior. After a few steps the responsibilities get more and more distinct, also leading to a more distinct posterior-distribution.

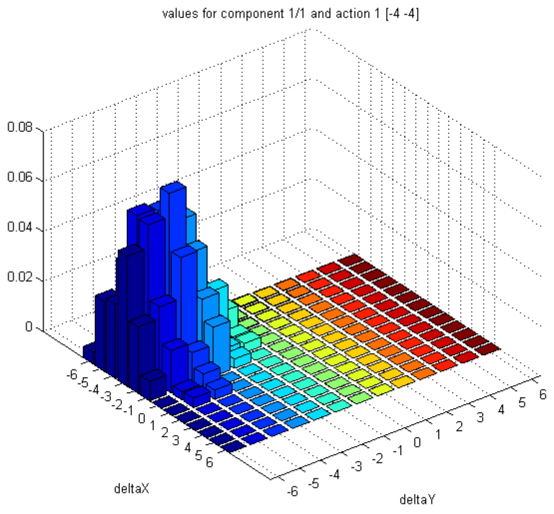
Figure 4.7 shows another example where 12 mixture components were used for the model. Again, the initial posterior (without any observations) reflects the uncertainty of the rotation angle of the new episode (Figure 4.7a). After the first step the clusters 6 and 8 seem to be most likely to have created the data for the new episode (see evolution of responsibilities in Figure 4.7d and the corresponding posterior in 4.7b). After three steps, cluster 11 has the highest responsibility, which is in this case correct since the angle of cluster 11 is closest to the angle of the new episode. But cluster 8 still has a responsibility of about 0.3 which can also be seen in the relatively "broad" posterior in Figure 4.7c. The initial responsibilities as well as the posteriors are based on very few observaitons. If these observations happen to be "quite off" the mean-rotation angle (remember that the actually taken velocity is drawn from a normal distribution around the mean rotation-angle), the wrong cluster might get the highest responsibility.



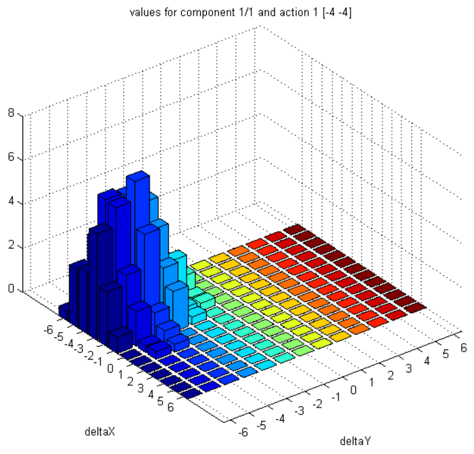
(a)



(b)



(c)



(d)

Figure 4.6: Evolution of the posterior distribution - (a) shows the initial posterior before taking a single step, (b) and (c) show the posterior after 6 and 9 steps respectively and (d) depicts the corresponding prior-component (with the highest responsibility).

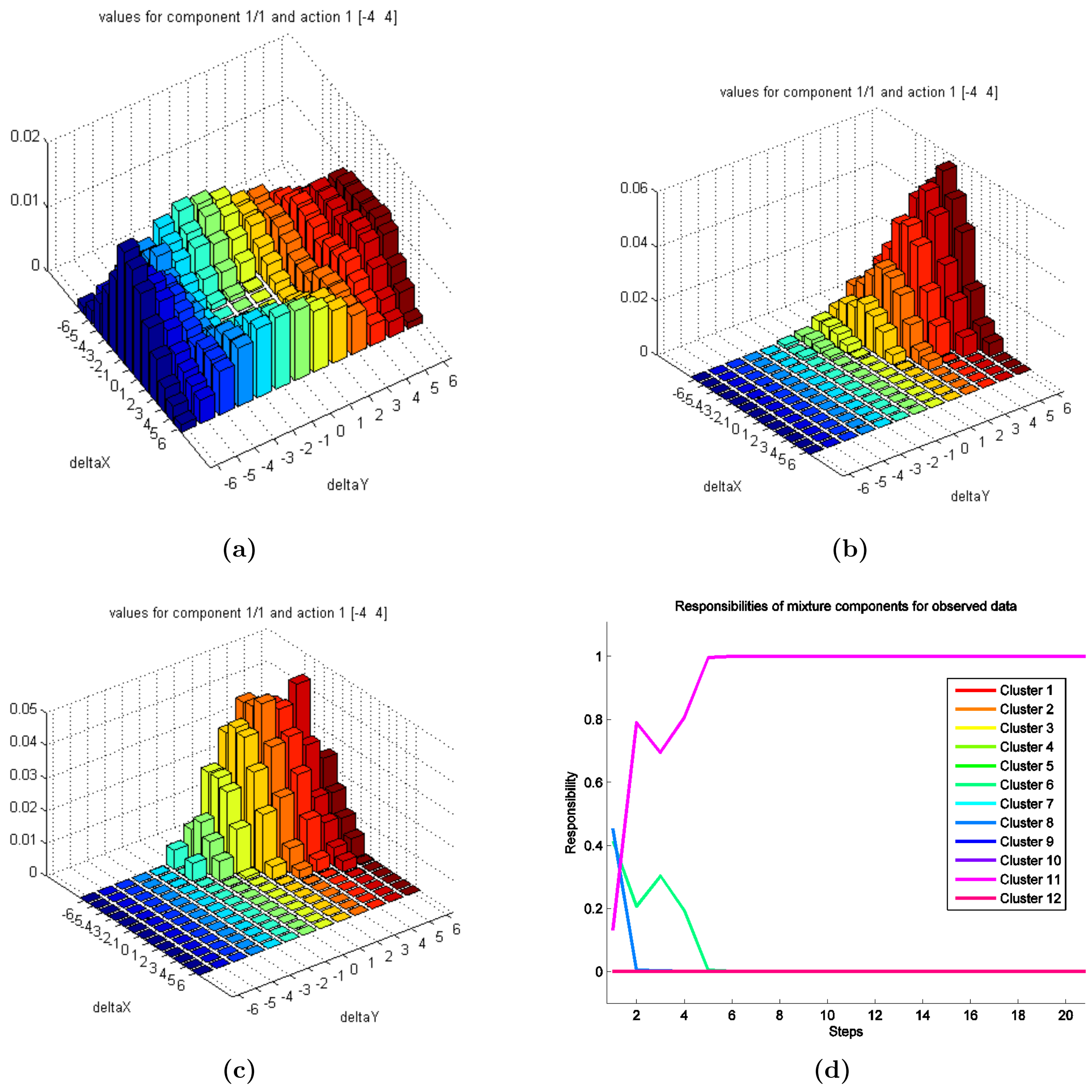
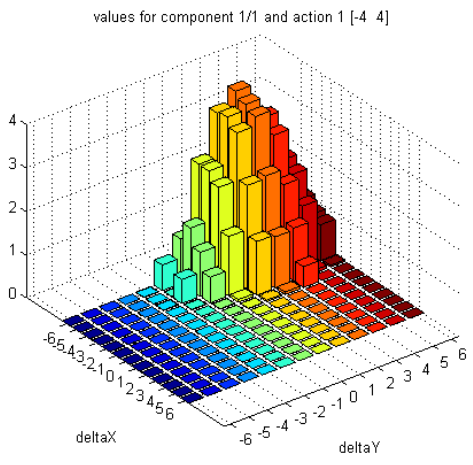
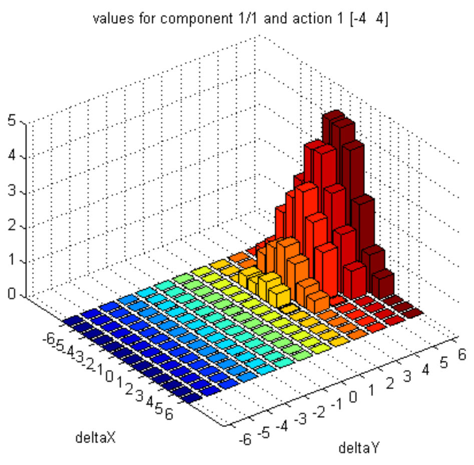


Figure 4.7: Evolution of the posterior (12 mixture components) - (a) shows the initial posterior before taking a single step, (b) illustrates the posterior after 1 step, with the wrong mixture components having a high responsibility. (c): Posterior after 3 steps, where the responsibility of cluster 11 is about 0.7 but cluster 8 also has a responsibility of about 0.3, resulting in a "broad" posterior. (d) depicts the corresponding evolution of the responsibilities.

Figure 4.8 shows the learned Dirichlet parameters for cluster 11 and 8 (compare them to the posteriors in Figure 4.7). Figure 4.9 illustrates the significant difference between the true distribution of the new dataset (Figure 4.9a) and the early distributions that are based on very few samples of the true distribution (Figure 4.9b). Notice how the ovservation data even after 50 steps is still quite sparse and does not have a similar shape to the true distribution - e.g. the "outliers" light blue at $\Delta y = -2$. Under this considerations it is even more remarkable how fast the responsibilities converge to the correct values (usually within the first five samples). The posterior in that early stage is then almost entirely based on the learned priors. After 50 steps the impact of the observation data on the posterior is already significant - see Figure 4.9c. However, in the experiments shown in the next section (Sec. 4.2), the agent usually performs between 15 to 25 steps to reach the goal position, i.e. its navigation strongly depends on the learned priors and the responsibilities, which in turn are computed from the observation data.

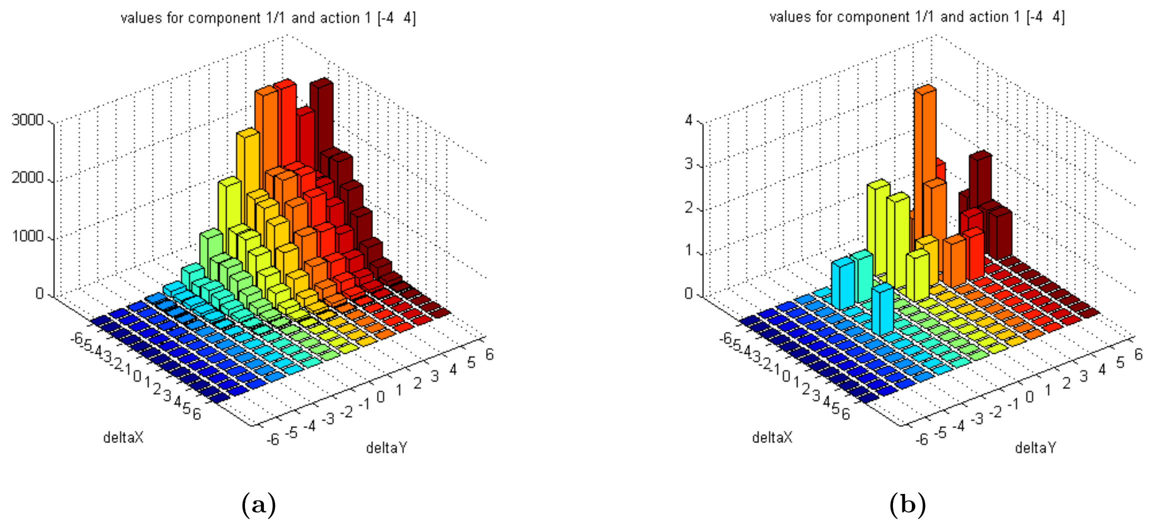


(a)



(b)

Figure 4.8: Dirichlet parameters (12 mixture components) - (a): Learned parameters for mixture component 11 (of 12). This mixture component will get the main-responsibility for the new dataset after four steps - see also Figure 4.7d. (b): Parameters for mixture component 8, which has a high responsibility after the first step.



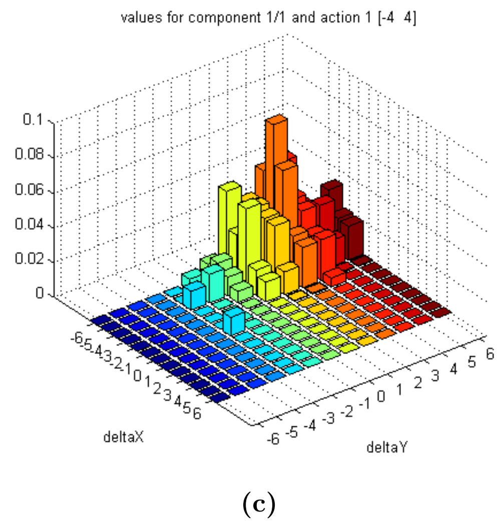


Figure 4.9: Multinomial parameters (12 mixture components) - (a): True multinomial parameters of the novel dataset. (b): Multinomial parameters after 50 steps - the responsibility-computation as well as the posterior are based on this distribution! (c): Corresponding posterior after 50 steps - the impact of the prior (Fig. 4.8a) is decreasing whereas the observation data (Fig. 4.9b) is starting to reflect in the posterior.

4.2 Advanced task

A description of the advanced task setup can be found in 3.4. Due to accelerations as actions, the corresponding planning problem becomes non-trivial and the simulation environment now actually consits of a gridworld where an agent tries to reach a certain goal position.

As in the previous task, a number of generated episodes is the basis for extracting the task structure (by fitting the model to the data, using EM). Afterwards a new episode is generated, with a novel angle for the rotation-distortion. The learned priors and the new data are then used to compute a posterior for the actually taken velocity difference, given the acceleration-command. The posterior-probability of ending up at a particular velocity change $\mathbf{v}' - \mathbf{v} = \Delta\mathbf{v}_l$ when taking action $\mathbf{a}_n$, given the priors $\boldsymbol{\alpha}$ and the hyper-priors $\boldsymbol{\kappa}$ as well as the observations of a single dataset D_j is given by:

$$P(\mathbf{v}' - \mathbf{v}|\mathbf{a}) = P(\Delta\mathbf{v}_l|\mathbf{a}_n,D_j,\boldsymbol{\alpha},\boldsymbol{\kappa}) = \sum_{k=1}^{K} q_j(k)\mathbb{E}\left\{\mathrm{Dir}(\boldsymbol{\alpha}_{k,n} + \mathbf{m}_{j,n})\right\} = \sum_{k=1}^{K} q_j(k)\frac{\alpha_{k,nl} + m_{j,nl}}{\alpha_{0k,n} + R_{j,n}}, \tag{4.3}$$

where $\alpha_{0k,n}$ is defined in (3.18) and $R_{j,n}$ is defined in (3.17).

The posterior and the transition-model finally form the state-transition-distribution (see Equation (3.25)), which is required for the planning-algorithm (value-iteration). Note that the posterior and thus the transition-distribution needs to be recomputed after every step.

A variation to the original task is the introduction of *delayed feedback*. In this setting, the likelihood of the data is updated with a certain delay, e.g. 3 steps. That means that the posterior, the responsibilities as well as the state-transition-distribution will be updated with the same delay as well. This behavior crudely resembles the concept of feedforward- vs. feedback-control which can be observed in biological motor control. In the first moments of sensorimotor control, the sensory system cannot provide any feedback (because it is still processing/propagating information); after a certain delay the feedback is used to adapt the movement accordingly - however the feedback will always have a certain delay. See the results in [2] of Braun et. al. for an illustration of this effect on a similar task with human test-subjects. Notice however, that the simulated robotic agent will get an immediate feedback of its new position on the grid-world (which is required by value iteration). Only the transition-distribution and all quantities based on it are updated with a delay. Thus the problem-setting concerned in this thesis can not be regarded as *pure* feedforward.

4.2.1 Experimental setup

The following results show illustrations for the movement of the agent in a simulated gridworld under varying task settings. If not stated otherwise the 70 datasets used for the initial learning stage were generated with random angles for the rotation-distortion, in the range of zero to 360 degrees. To introduce a certain bias towards the 0°-angle, more datasets were generated, using this (mean-) value with a variance of about 5.5°. This bias is naturally motivated and reflects a prior belief towards undistorted movements. Furthermore, it is no longer necessary to have a non-deterministic action selection (see A.2.2), since the bias will ensure that one certain action is expected to lead to a higher cumulative reward.

The gridworld has a size of 30x30 cells; possible velocities are $[-3,-2,-1,0,1,2,3]$ for each axis and possible accelerations are $[-2,-1,0,1,2]$ for each axis, resulting in $30^2 \cdot 7^2 = 44100$ different states and $5^2 = 25$ different actions. In the planning stage value iteration needs to consider every possible action for every possible state, leading to $44100 \cdot 25 = 1102500$ different state-action pairs. . In the worst case this requires at least two steps to decelerate to zero or change direction.

Heuristic value iteration

In order to be able to gain meaningful results, it is not sufficient to simply investigate single trajectories of the agent - eventhough the agent's commands are deterministic, the corresponding actions as well as the movement on the gridworld are not. It is therefore required to derive results that arise from many trajectories. On one hand this will be mean-speed profiles as well as the mean-distance to the target, where the mean is taken over all trajectories of a single trial. On the other hand, the plots in the following sections will show the individual trajectories, plotted into an occupation-probability map for each cell of the gridworld. The map gives a measure on how often the agent has visited a certain grid-cell during the whole trial (i.e. the rollout of several trajectories).

The computational demands for the given setup are quite high. The rollout of several trajectories is problematic and will lead to simulation-runtimes on the order of days - mainly due to value iteration (needs to be performed after every single step of every trajectory). To overcome this problem, a heuristic version of planning with value iteration is used: instead of performing a value iteration step after every step of the agent (which is required because the posterior changes due to the new observation), value iteration is performed only once *for every mixture component*. This corresponds to "computing a *plan* for every learned rotation-cluster". According to the current observations, the plan from the mixture component with the highest responsibility is selected and used to plan the subsequent step. In other words: Initially, a set of plans is computed and based on the current observations one of these plans is used to select the action for the next step. In this case, value iteration is based purely on the prior-knowledge and the posterior is not used for planning. Instead the observations are used to compute the responsibilities of the mixture-components for the current rollout.

On a first glance, this might seem like a severe simplification. But during the first, say 20 to 30, steps the observation data has very little impact on the posterior - in fact, it is almost similar to the corresponding prior. The reason why the KL-divergence between the posterior and the true distribution decreases rapidly during the first steps is, that the **responsibilities** of the mixture-components converge towards "good" values - i.e. one mixture component gets a very high (close to one) responsibility (whereas the initial responsibilities are equally distributed along all components; with a bias for the zero-degree component). Depending on the number of velocities and actions, it probably takes more than 100 steps for the observation-data to have a significant impact on the posterior - during the first steps, the priors dominate the posterior.

Therefore it is valid to use the *heuristic value iteration* - in fact, experiments with single trajectories show that there is almost no difference between the two versions as far as the qualities of the resulting trajectories are concerned. As far as the computational demand is regarded, the heuristic version leads to major improvements: if the value iteration plans have been computed for every mixture component (once), they can be used to perform trials with dozens of trajectories in a few minutes. The (small) error introduced by not using the true posterior is *significantly outweighted* by the ability to perform experiments with dozens of rollouts and to draw statistically "stable" results from them (by using averaging measures).

4.2.2 Trajectories under different rotation-distortions

Figure 4.10 shows the results of a trial with 50 rollouts. The angle of the rotation-distortion has been set to 90° and the feedback delay has been chosen to be three steps. In the occupancy map, you can see the 90° distortion with the naked eye - this is due to the initial bias of the mixture component for zero degrees. Without any feedback, the agent assumes a rotation distortion of 0° and executes the corresponding actions - therefore the true angle of the rotation-distortion becomes visible.

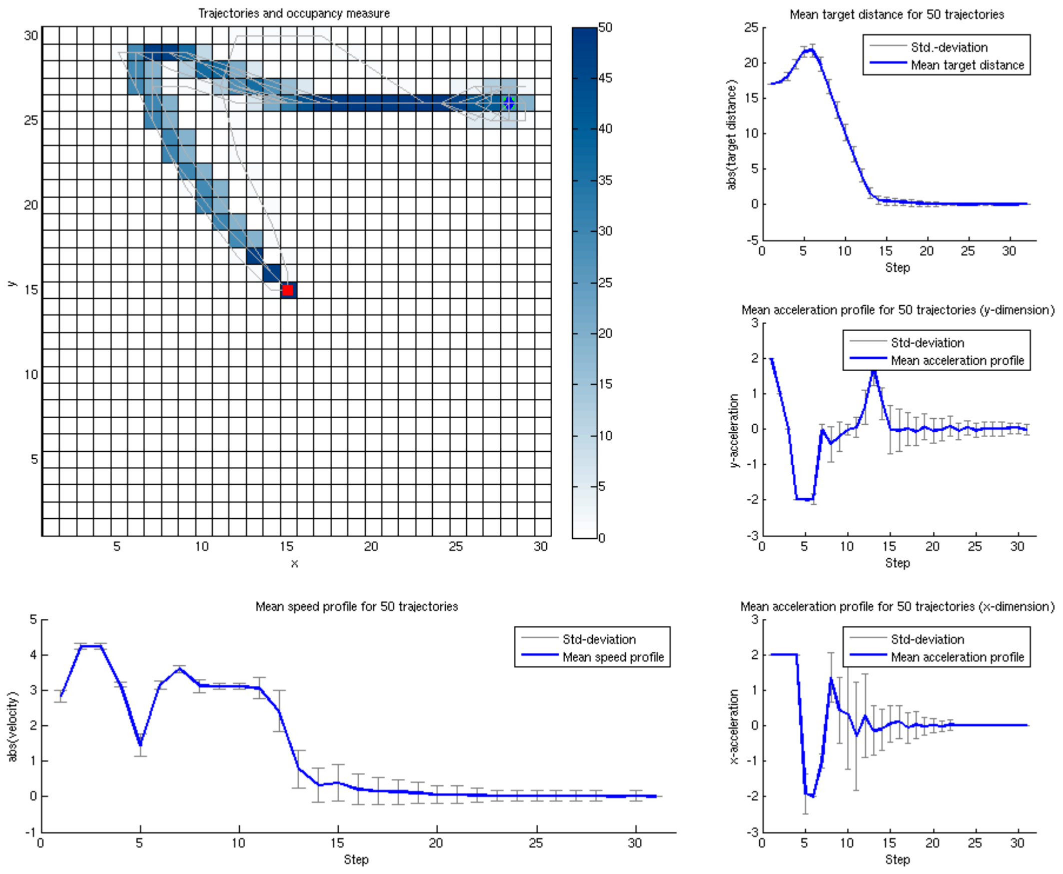


Figure 4.10: Trajectories for 90° rotation-distortion - trajectories (light gray) and occupancy map in the top-left plot. Initial agent state in red and goal state in green - final position of agent given by blue diamond. Mean speed of the agent in the bottom-left and mean target distance in the top-right plot. Mean accelerations in the center-right plot (y-axis) and bottom-right plot (x-axis) - notice that there is a rotation of 90° between the acceleration axes and the gridworld/velocity axes. The rotation distortion of 90° can be seen with the naked eye, as well as the feedback delay of three steps (see speed profile plot).

Also notice the mean speed profile in 4.10 - after the feedback delay of three steps, the agent "realizes" the rotation distortion and starts decelerating, which takes two subsequent actions, and then starts going towards the correct direction. To be more precise: the observation data is updated with a delay of three steps, which means that the responsibility for the correct cluster will become significant only after these three steps. Compare this speed profile with the results of Braun et. al. (see Sec. 2.1.2) - who also show two significant peaks in the recorded speed profiles, denoting the feedforward- and feedback-phase of human visuomotor control.

With the defined transition-model it is actually "quite hard" for the agent to reach the goal state and stay there, because there is a quite high chance of dropping out of the goal state when decelerating in it. This can lead to an oscillatory movement around the goal state - at least for a few steps. However, when using a different transition model, a different time step or simply a larger action-set this problem vanishes.

Figure 4.11 illustrates this "oscillatory" effect - it has been generated, using a rotation-distortion angle of 240° and a feedback delay of two steps. The model, in this case, uses seven mixture components and it seems as if the component responsible for 240° is still a bit too coarse. Due to the small number of mixture components, some averaging effects are still observable and planning becomes brittle when using these priors - especially when trying to decelerate into the goal-state and stay there. The agent often breaks too early or "overshoots" the goal-state, which can also be seen by the range of the occupancy map - eventhough only 50 trajectories were used, there are cells that have been visited a lot more often, especially the ones directly adjacent to the goal as well as the goal-position itself. Also, the number of steps (on the x-axis of the two bottom plots) is a lot higher than in the previous case. A sufficient number of mixture components avoids such problems to a large extent.

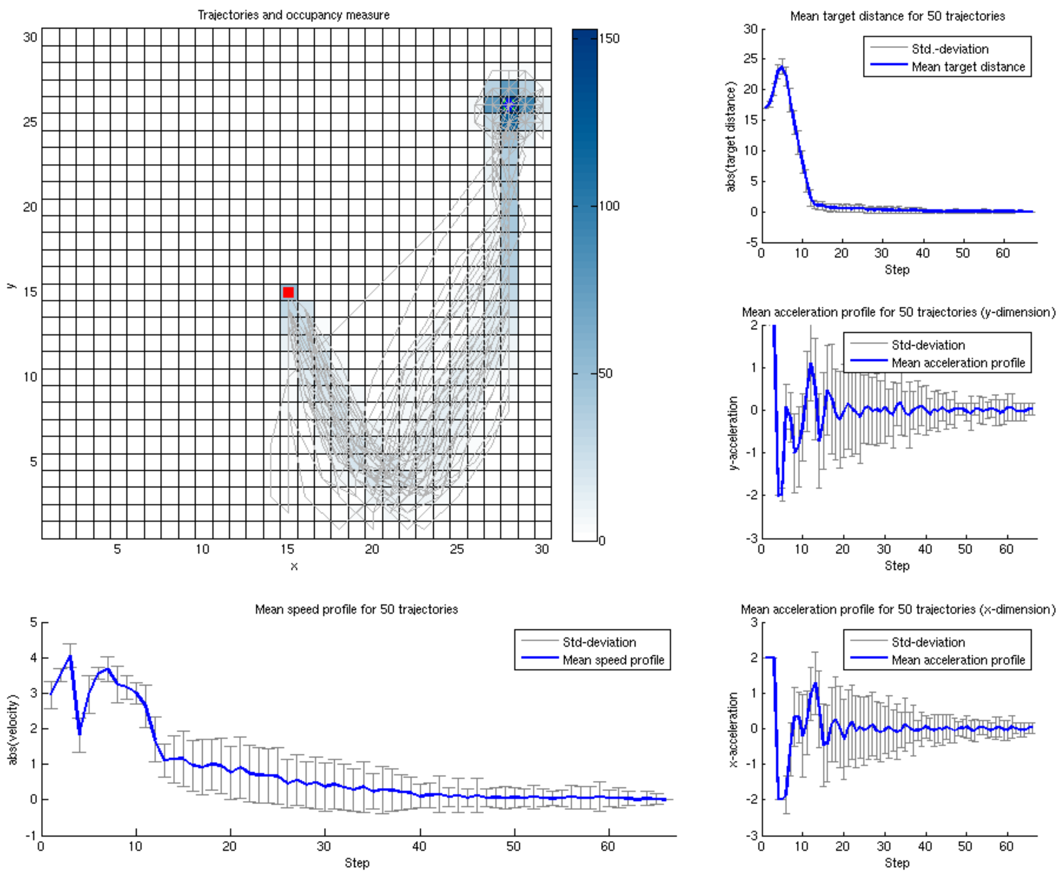


Figure 4.11: Trajectories for 240° rotation-distortion - trajectories (light gray) and occupancy map in the top-left plot. Initial agent state in red and goal state in green - final position of agent given by blue diamond. Mean speed profile: bottom-left and mean target distance: top-right plot. Acceleration profiles in the center-right (y-axis) and bottom-right plot (x-axis). Rotation distortion of 240° between the acceleration axes and the gridworld/velocity axes. The model was fitted to data, using seven mixture components - which is not sufficient and averaging-effects are still visible. The mixture-component responsible for the 240° is still too coarse and which leads to a "suboptimal plan". The effect is particularly severe around the goal-state.

4.2.3 Trajectories for various feedback delays

Figure 4.12 illustrates the agent's trajectories for a feedback delays of two steps, whereas Figure 4.13 shows the trajectories with no delay (zero steps). The rotation-angle for both trials was chosen to be 90°, which allows for comparison with Figure 4.10.

The main-result of this section is that the variation of the feedback delay leads to the expected behavior in the agent's trajectory, i.e. the agent starts off into the wrong direction and then "realizes" its mistake with a certain delay and then compensates the rotation distortion. The

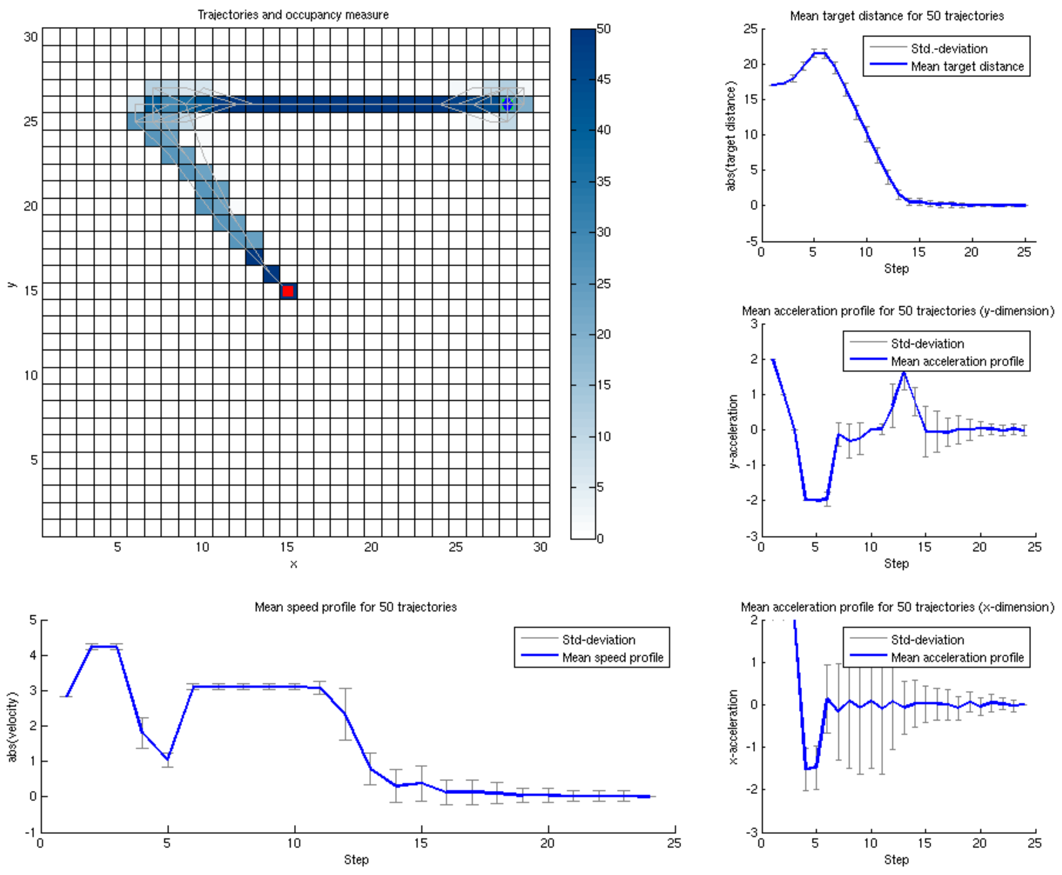


Figure 4.12: Trajectories for a feedback delay of two steps. Initial agent state in red and goal state in green - final position of agent given by blue diamond. 90° rotation distortion between acceleration and velocity.

larger the feedback deleay, the longer the it takes for the agent to start compensating the rotaion. While this might not provide large novel insights, it certainly confirms expected qualities of the model or rather the methodology. It is worth noticing that the number of feedback delay steps directly corresponds to the first local minimum in the mean speed profiles (with an additional two steps for decelerating). The experiment with no feedback delay (Fig. 4.13) shows that the mean target distance is (almost) immediately decreased and the corresponding speed profile does not disply the significant twin-peaks of feedforward- vs. feedback-control. The speed profile in Figure 4.12 (with two steps of feedback delay) shows a plateau rather than a second peak, this is due to the maximum velocity of 3 units (the reason why the initial peak is higher is that the velocity during that timestep was saturated on both axes, while at the plateau-phase it is zero on the y-axis).

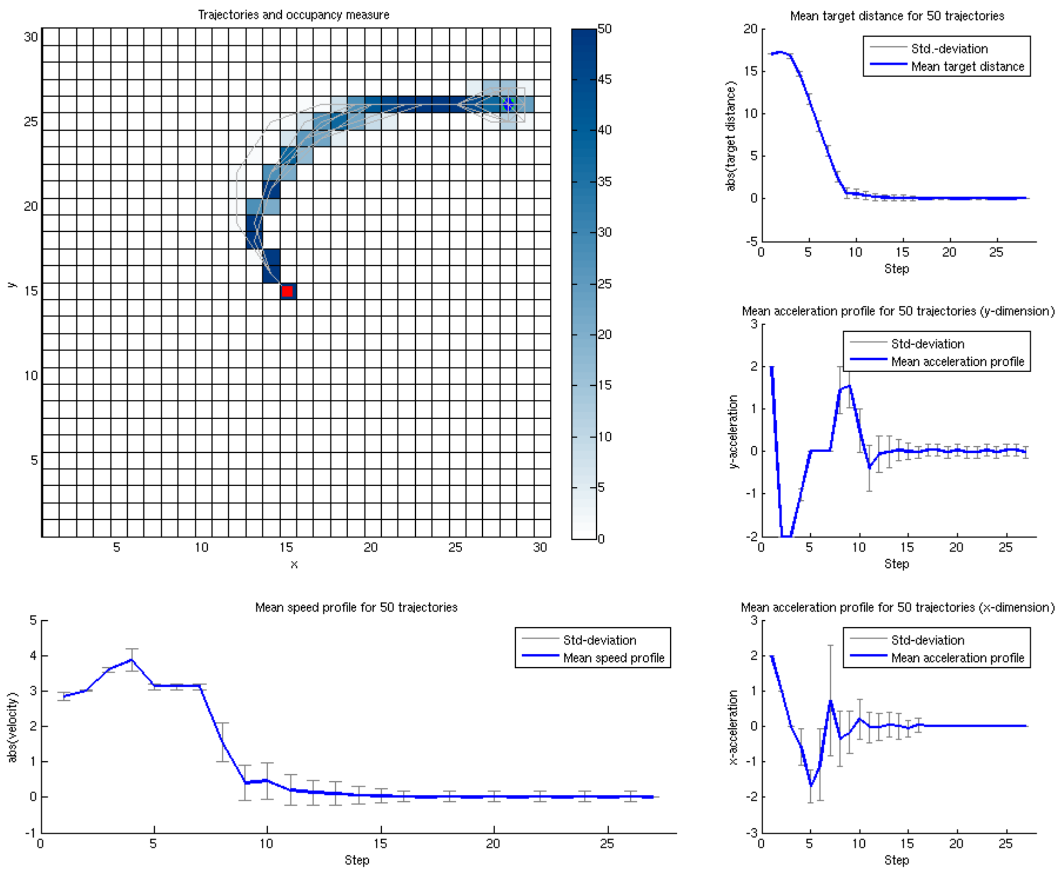


Figure 4.13: Trajectories with no feedback delay. Initial agent state in red and goal state in green - final position of agent given by blue diamond. 90° rotation distortion between acceleration and velocity.

4.2.4 Trajectories with suboptimal action-selection

Under certain circumstances, it might occur that the learned priors all have exactly the same weight, i.e. they all are responsible for the same number of mixture components. E.g. when using only four angles (0°, 90°, 180°, 270°) for generating the learning sets. If the agent further has an unfavorable initial position, it might be the "best" action to simply remain in this position (by applying an acceleration of (0,0)). Because initially there is no observation data, the responsibilities of all mixture components would have the exact same value. A bad choice of action could therefore move the agent further away from the goal-state, resulting in a penalty (increased negative reward). However, taking action (0,0) does not allow the agent to infer the rotation-distortion, since this action is ambiguous for all rotation-angles ("By not moving at all, how could the agent infer the rotation-angle?"). It might therefore be desirable to have a non-deterministic selection rule for taking suboptimal actions "once in a while". The two options implemented, to achieve such an action selection mechanism are ϵ-greedy and softmax action selection See the appendix A.2.2 for more details.

The following results show the effects of different temperatures for the softmax action-selection mechanism exemplarily on two trials - in Figure 4.14 the softmax-temperature was set to 0.0002 (low temperature), whereas Figure 4.15 presents the results using a temperature of 0.001 (high temperature).

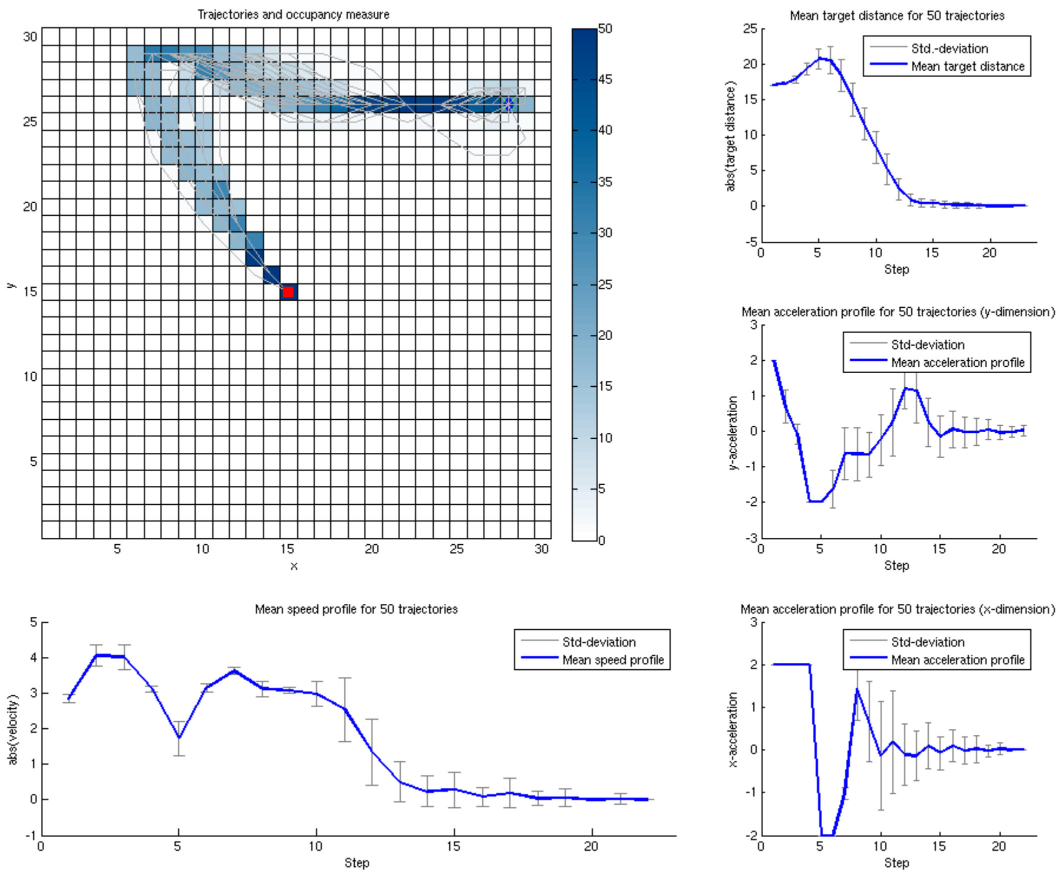


Figure 4.14: Trajectories with softmax action-selection - 90° rotation distortion, three steps feedback delay. Initial agent state in red and goal state in green - final position of agent given by blue diamond. Softmax temperature was set to 0.0002

Notice that these results are illustrative demonstrations - the implementation with the heuristic value iteration as well as the (naturally inspired) bias in the priors towards zero-degree distortions do not require such an "explorative" action-selection strategy. Nonetheless, the results also underline the robustness of the implemented system - eventhough the action-selection is far from optimal, the mean-speed profile as well as the mean-target distance is not severely worse

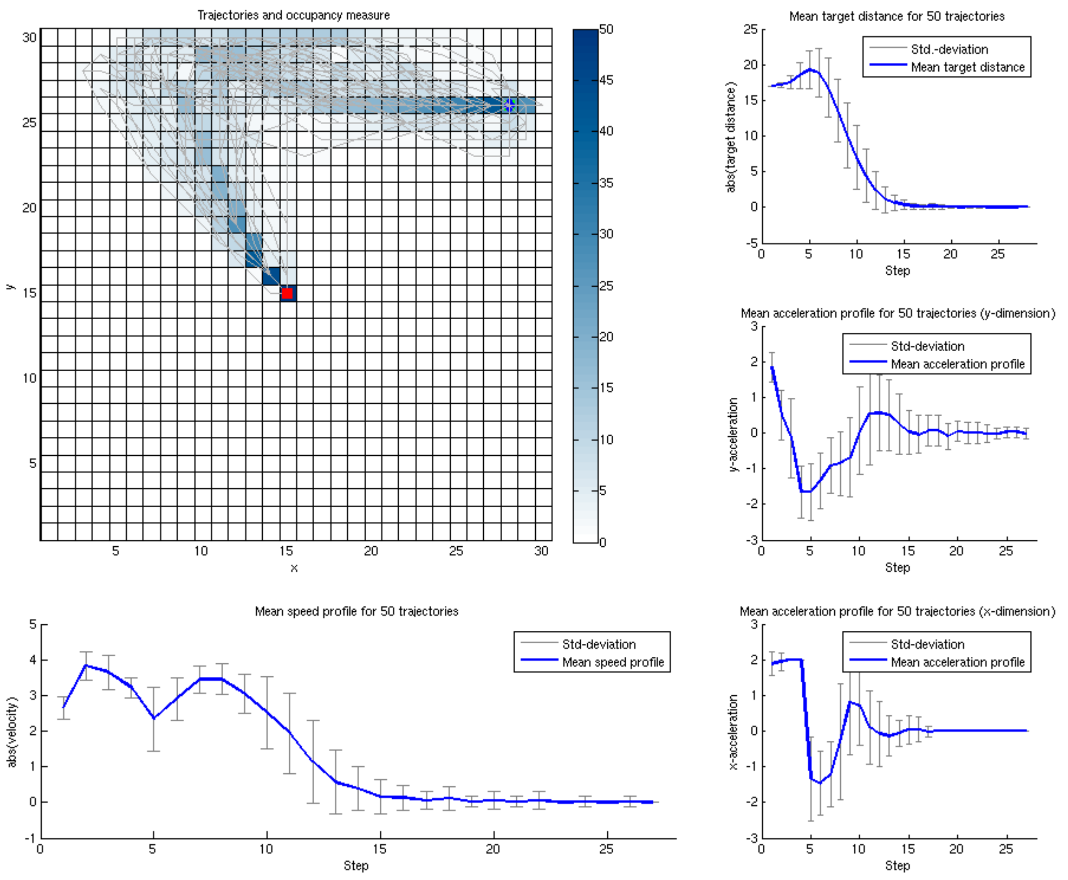


Figure 4.15: Trajectories with softmax action-selection - 90° rotation distortion, three steps feedback delay. Initial agent state in red and goal state in green - final position of agent given by blue diamond. Softmax temperature was set to 0.001

than in the deterministic-action-selection case (Figure 4.10). Also the plots help in understanding the difference between the movement-noise (the application of acceleration-commands as well as the movement within the gridworld are noisy) and differences in the trajectories caused by suboptimal actions.

ϵ-greedy action selection can be used to produce similar results - however it is hard to compare both methods against each other or to gain any quantitative results from the simulations. Since the plots look quite similar from a qualitative point of view - they have been ommitted. From an implementational point of view, the theory can be confirmed: Specifying the ϵ-parameter was simple and intuitive (percentage of random actions), while the selection of a "good" softmax-temperature was more or less an empiric process which led to satisfying results for the given problem-setup but fails for other problem-tasks (with smaller/larger action- or state-sets).

4.2.5 Illustrations of the value function

The value function provides a measure on the expected return (expected cumulative reward) for every state within the statespace. Its shape strongly depends on the reward-function (see 3.4.2). In case of a discrete setting, the value function is usually represented in a tabular form. The result of value iteration is the *action-value function* on one hand (which indicates the expected return of a state when taking a certain action), as well as the *state-value function*, or simply value function, on the other hand. The value function consits of the expected reward when always taking the "best" action (i.e. with the highest action-value) - thus the state-value function is the maximum of the action-value function for every state in statespace.

To get a more intuitive insight, consider Figure 4.16: it shows a subset of the learned value function. The statespace is formed by all possible gridworld-positions and all possible velocities. Figure 4.16 shows the state values for all positions but at a *single* velocity (in this case (0,0)). The values have been mapped onto the corresponding gridworld-positions. It is easy to see that the function has a global maximum at the goal state (to generate these plots the goal been set to (15,20)) and the penalty increases with the distance from the goal state, as expected (see Eq. (3.26) for the definition of the reward function).

Figure 4.17 illustrates the subset of the value function for all states with velocity $(-3, -3)$. Compared to Figure 4.16, bottom-left area of the gridworld now has a significantly decreased expected return (notice the magnitude on the z-axis, which has almost doubled). On the other hand, the very upper-right area has an increased expected return. This is correct, since velocity $(-3, -3)$ will lead towards the goal-position, if the agent is in the upper-right corner of the

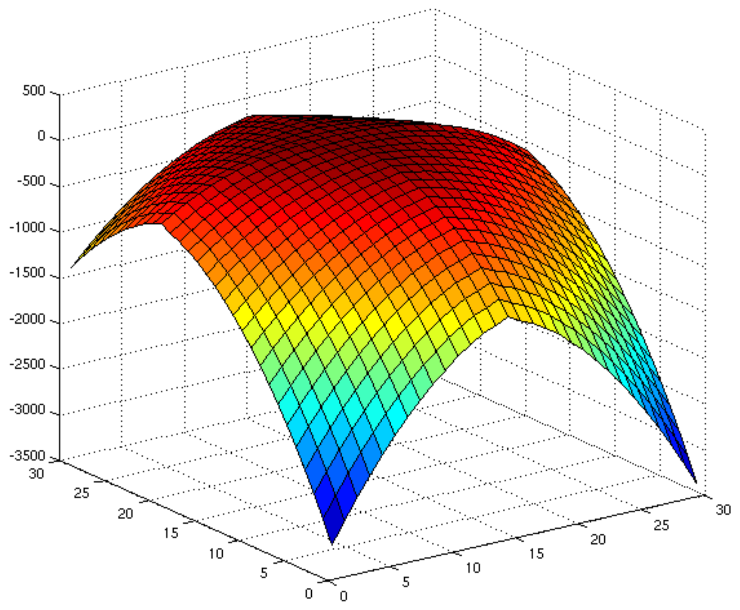


Figure 4.16: Value function for velocity (0,0)

gridworld, whereas in the bottom-left area, it will drive the agent further away from the goal. The goal-position does not have a very distinctive maximum, but rather sits on a plateau - this is also correct, as the velocity is too high to be able to "halt" within one timestep. Thus, other states (at the same position with lower velocities) have a higher expected reward at the goal position. Further, the value function has some sort of fold on the left and lower border - this is the result of the convention, that the agent cannot leave the gridworld, but will simply bump into walls without changing its velocity.

Other than that, it is hard to make further qualitative comments on the value function, since the "shape" of the function does not significantly change - it is rather the numerical values that are important. The same also applies to the action-value function, which would show the "best" action for the states of the problem task.

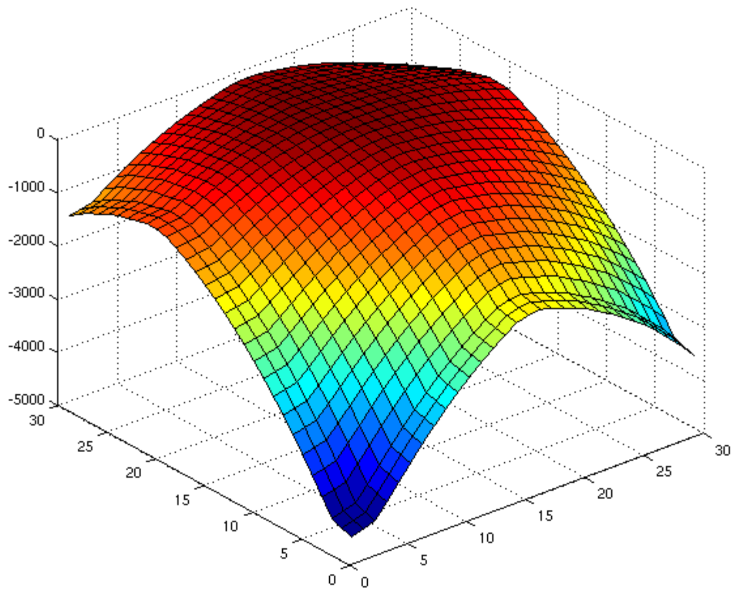


Figure 4.17: Value function for velocity (-3,-3)

4.3 Learning stage - parameter fitting

This section presents the results for the optimization of the hyper-prior parameters during learning stage. The results have been acquired by performing (generalized) EM over the course of 300 iterations, using a database of 50 datasets with random angles for the rotation-distortion. The action-set contained discrete actions in the range of $(-4{,}4)$ on each axis, the corresponding state-set (velocity-changes or velocities, depending on the problem-task) allows discrete values in the range of $(-6{,}6)$.

The results in Figure 4.18 show the evolution of the hyper-parameters of both Beta-mixture components ($\boldsymbol{\lambda}$). All values have been initialized to 1, which corresponds to a uniform distribution. As the plots show, the parameters converge to certain values during the course of the learning stage. Figure 4.19 illustrates the corresponding PDFs - for both mixture components with the finally learned parameters. First of all, the PDFs are clearly different eventhough both Beta-distributions were initialized with the same parameters. Furthermore, one of the distributions seems to be *responsible* for values between 0.02 and 0.5. These values correspond to the range of values for (significantly) non-zero, *normalized* $\boldsymbol{\alpha}$-values (see Sec. 3.3.1). The other Beta-distribution has been fitted to very small values, close to being zero. Of course these second distribution captures the $\boldsymbol{\alpha}$-values at the lower limit - which was 0.001 in this case (since values very close to zero cause numerical problems). The striking result of these plots is that the optimization has led to parameter-settings for two clearly different PDFs that are intuitively reasonable - one mixture component has been fitted to "explain" the significant non-zero values whereas the other component is responsible for the values at the lower-bound (most of the $\boldsymbol{\alpha}$-values usually are at the lower bound - see Figure 4.3 for comparison).

Notice that one of the parameters (b2) is driven toward saturation. This is due to the optimization trying to make the corresponding Beta-distribution even "sharper", in order to fit it even more on the lower-bound $\boldsymbol{\alpha}$-values. However, too large values would cause numerical problems, thus the hard limit at 2000.

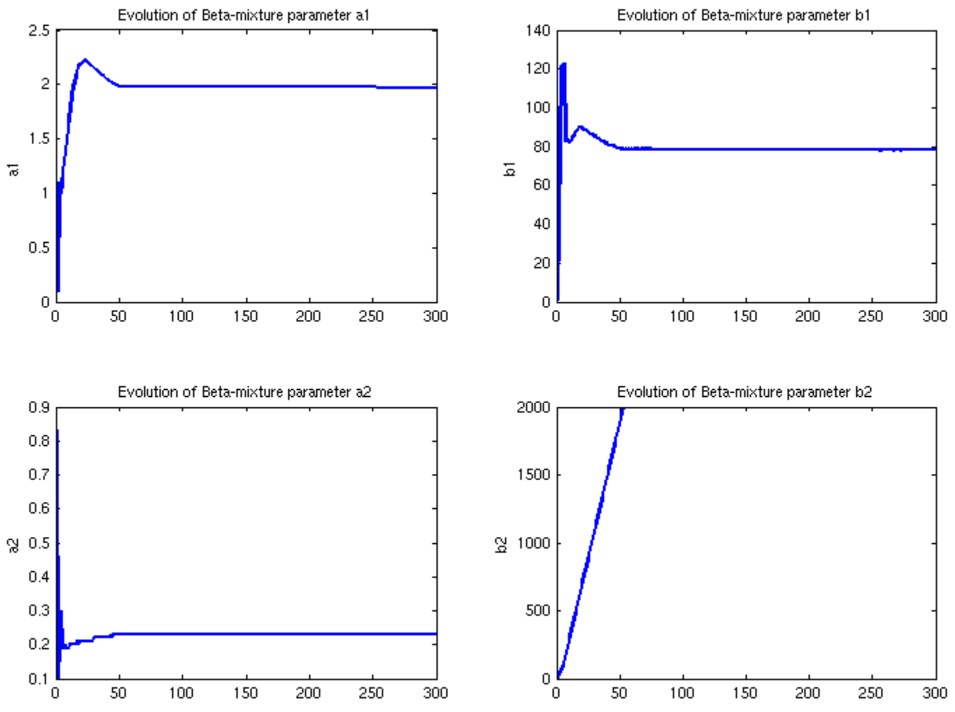


Figure 4.18: Evolution of parameters of Beta-mixture - all values converge - the b2 parameter is driven towards saturation.

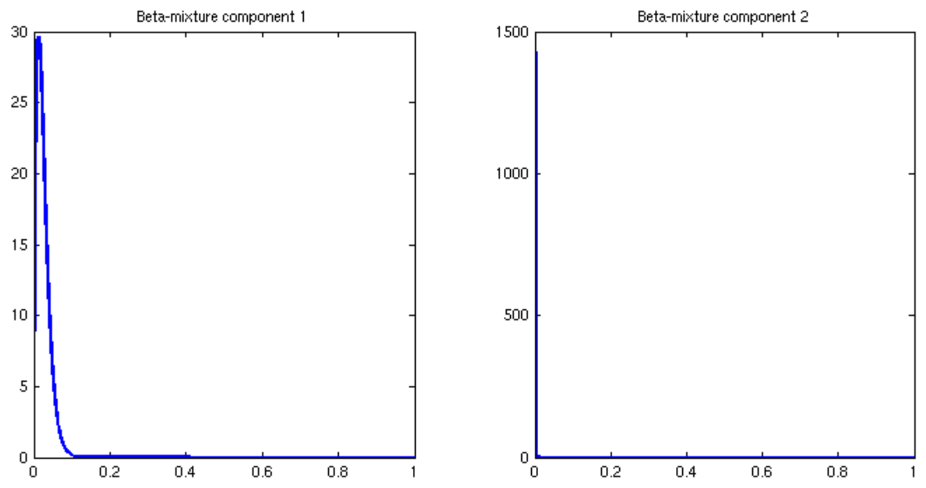


Figure 4.19: PDFs of Beta-mixture components - two clearly different distributions - explaining the significant non-zero values of $\boldsymbol{\alpha}$ as well as the values at the lower bound (0.001).

The weights of the Beta-mixture ($\mathbf{x}$) are optimized as well and the results are shown in Figure 4.20. The weight-values converge nicely to about 0.64 and 0.36 which corresponds to the ratio of significant non-zero values versus values at the lower bound of $\boldsymbol{\alpha}$.

Summing up, the optimization has started from initial values with the same parameters (uniform distributions) and almost equal weights (0.5 with some slight noise). For all parameter-values the optimization has converged and the resulting distributions explain both "kinds" of $\boldsymbol{\alpha}$-values whereas the weight of the distributions corresponds to the ratio of these value-kinds.

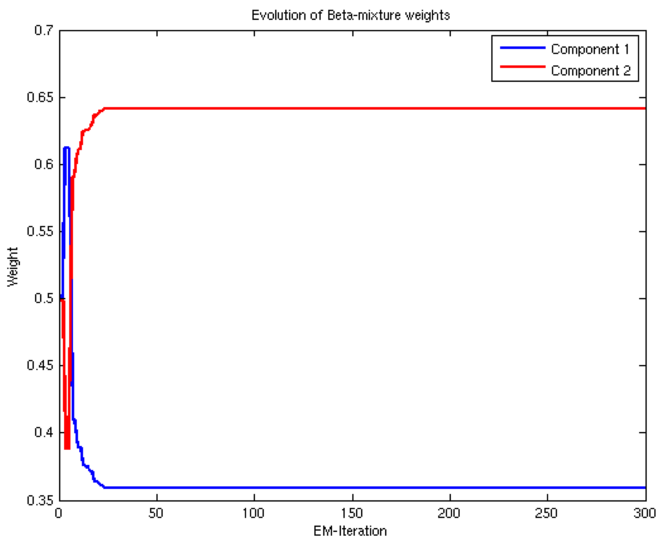


Figure 4.20: Evolution of Beta-mixture weights - the weights correspond to the distribution of significant non-zero values of $\boldsymbol{\alpha}$ as versus values at the lower bound (0.001).

Furthermore, the distributions and the weights are consistent with intuitive beliefs on the shape and responsibility of the two distributions. While this might not be a pure justification for introducing the hyper-prior distributions, they seem to be a reasonable extension to the basic model that can be nicely fit to the observation data. They further introduce some advantages (limitation of the $\boldsymbol{\alpha}$-values only needed at the lower bound) and are also defensible from a mathematical point of view (extending the model towards a full Bayesian model). By using these hyper-priors, the likelihood of test-datasets could be slightly increased, compared to the basic model, however these results have to be interpreted with care since there are several other factors (such as the limitation of certain values) involved as well.

Figure 4.21 shows the evolution of the Gaussian parameters during the course of EM-iterations. Initially, the randomly initialized $\boldsymbol{\alpha}$-values are driven towards zero which also reflects in the Gaussian parameters (maybe even the early parameter of the Beta-priors have an influence on that). But then they converge towards values that match learned $\boldsymbol{\alpha}$-priors. Degenerate cases and outliers could be significantly reduced - the Gaussian prior drives the Dirichlet-priors to be "more similar", which was the intended goal.

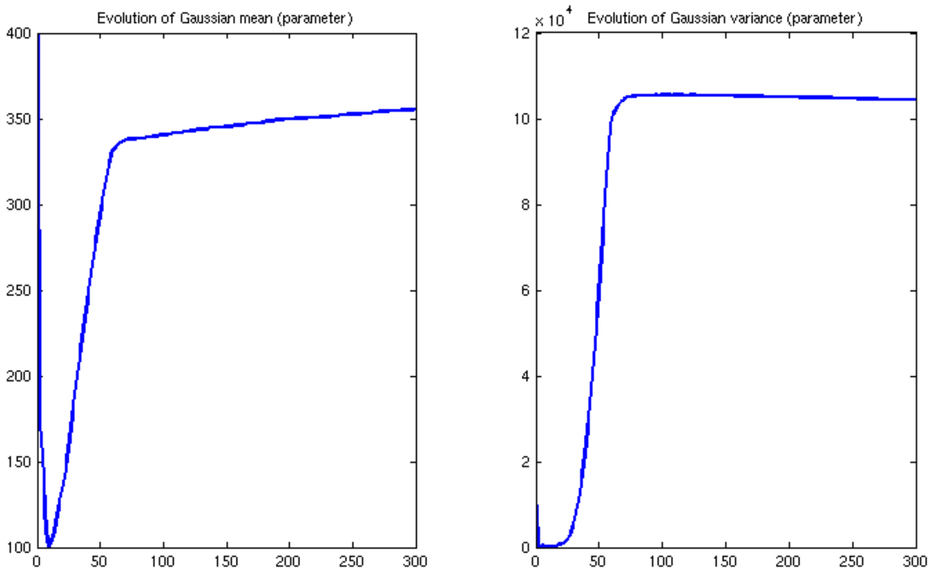


Figure 4.21: Evolution of parameters of the Gaussian-prior

As a final result, Figure 4.22 shows the evolution of the log-likelihood during the learning stage, which is to be optimized by adapting the model parameters using expectation maximization. As the plot illustrates, the log-likelihood converges nicely - for this particular case 300 iterations are sufficient (this has also been validated with another measure, since the plot shows values that range over a large magnitude and a simple "optical inspection" would not be sufficient).

The results in this section show an illustrative case - there are other cases where the fitting of the hyper-prior parameters did not lead to such significant results. Especially with "small"

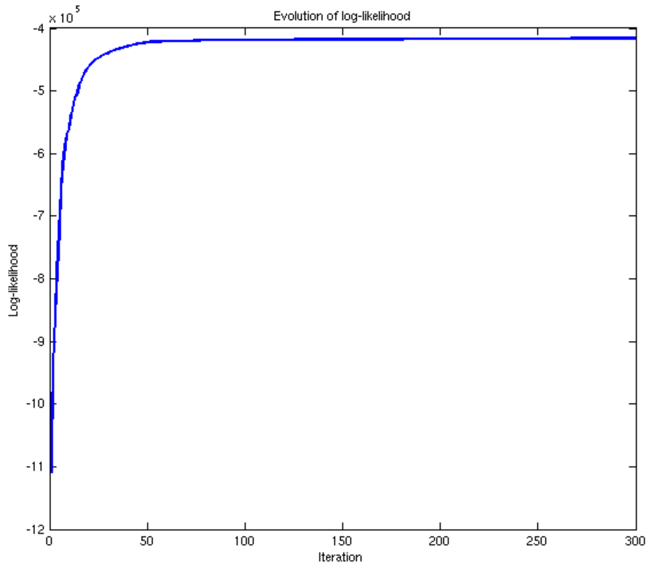


Figure 4.22: Convergence of log-likelihood during learning-stage.

problem-setups, consisting of rather small action- and state-sets the difference between the Beta-mixtures is less distinctive. Most of these issues are related to the overall learning-parameters such as learning-rates, step-sizes and limiting-values. With enough fine-tuning, the model could probably perform a lot better in these suboptimal cases. However, this fine-tuning can turn into an elaborate process and in many cases it is hard to quantitatively evaluate the impact of certain measures - more on this in the next chapter.

5 Discussion and conclusions

This chapter reviews the initial problem statement and reflects on the results gained through the experiments in the implemented simulation environment. The first section discusses the findings in terms of the thesis' objectives with a focus on the fast-learning aspects. The subsequent section discusses known issues of the implementation as well as the approach in general.

The third section provides some inspiration on potential future research and the final section of the chapter concludes the thesis in a brief summary.

5.1 Research objectives

The main-goal of this thesis, was to demonstrate the application of the proposed model for the given problem task and show that the model is capable of extracting structural invariants and exploit them to rapidly adapt to novel but similar task-instances. This goal has certainly been reached and the results for the *fast-learning* capabilities are positive. Consider the KL-divergence analysis (Sec. 4.1.1): compared to a model that does not use any priors, all models with prior-knowledge are able to faster provide a posterior-distribution of "better quality", requiring less steps (or data-points/observations). This is still valid for models with very few mixture components, eventhough they perform quite poor, compared to models with a sufficient number of mixture components.

A more intuitive illustration of fast learning is shown with the results of the advanced task (see Sec. 4.2.1). The (mean) trajectories of the agent under different rotation-distortions show that the agent is capable of rapidly adapting to the new task (of course, depending on the number of feedback-delay steps) and compensating for the rotation-distortion. If the model has gained sufficient experience (in the learning-stage), it only takes a handfull of observations to adapt to a novel task-instance.

The plots in Section 4.1.2 provide insight on the structure of the task and the way this structure is extracted and captured by the model. They show the learned parameters for some of the Dirichlet-mixture components which can be interpreted as a measure for the probability of ending up at a certain velocity/acceleration when executing the corresponding command. The evolution of the posterior illustrates the evolution of the agent's belief of this transition-distribution measure over the first steps of a novel episode and nicely shows the initial uncertainty that rapidly vanishes as the number of observations grows.

These results underline the model's capability of extracting structural invariants of the problem task and using them to rapidly adapt to new, but similar tasks, which was the main research-interest of this thesis. From a conceptual point of view, the model is capable of "learning to learn", or to be more accurate *learning to plan*, and this capability is highlighted in contrast to planning only, without using any prior-knowledge.

The research-scope of this thesis was to provide a “proof-of-concept” implementation of the proposed model and and use it to reflect on the fast-learning capabilities. As such, the implementation is not intended to be used for further developments (unlike a prototype-implementation). However, some parts of the implemented simulations might be of interest in future projects. Also, the implementation can be used for comparing it against different methods.

5.2 Shortcomings and possible improvements

Besides the positive results in terms of applying the proposed approach to the problem task and illustrating its fast-learning capabilities, there are a few issues that might be of concern - most of them deal with implementation-details but some also regard conceptual aspects.

5.2.1 Expectation maximization

The expectation maximization algorithm is commonly used in various applications (pattern recognition, data mining, machine learning) to determine the parameters of statistical models in a maximum likelihood approach. It has been used for decades now (originally proposed in [21]) and there are various kinds of implementations as well as extensions and methods based on EM (see [29]). Another advantage is that it has solid theoretical foundations and at least for analytically tractable models it will usually allow fast implementations (with comparatively low computational demands).

However, there are a few shortcomings known to EM (or the maximum likelihood approach in general). For a detailed discussion on this issue as well as illustrative examples see Bishop [14] chap. 9 and 10. Especially in conjunction with mixture models, EM can become problematic. On one hand it can easily lead to degenerate solutions, where a single mixture component collapses onto a single data-point/observation (perhaps even an outlier). Once a mixture component has lost the responsibility for a larger number of data-points and its parameters have been fitted to a few data-points, it is almost impossible for EM to recover from such a scenario (in some special cases this might actually be the desired behavior, though). Furthermore, EM requires certain analytical calculations, which can quickly become infeasible or don’t yield closed-form solutions for models of practical interest. Of course, there are workarounds but they will increase the computational demands.

The framework of *variational inference* could overcome some of these inherent problems. For mixture models, it is capable of inferring the number of required mixture components from the given data (which is a great problem for EM, see 5.2.3). To be able to apply variational inference, the parameters of the model ($\boldsymbol{\alpha}$-values) need to be absorbed into the set of latent variables and prior-distributions for these parameters are needed. The extended model used in this thesis would more or less fulfill such a requirement. The computational demands, for applying variational inference should not be much higher, compared to EM. The reason why EM was also used for the extended model is that it worked quite well (using the heuristic methods such as copying degenerate mixture components, etc.) and the existing implementation for the basic model could simply be extended. The implementation of a variational approach would have been more time-consuming and furthermore the method for fitting the model to the given datasets is out of scope for this thesis.

5.2.2 Value iteration

Value iteration is a simple, yet powerful algorithm for solving the Bellman (optimality) equation. As such, it seems perfectly suited for the planning problem in the given problem task. One major drawback of value iteration is that it needs a full model of the environment - in this case the full state-transition distribution (i.e. the probability of ending up at a certain position with a certain velocity when being at another position with another velocity and taking a specific action - unfortunately this quantity is needed for every possible combination of positions, velocities and actions). As expected, the computational demand for the full transition distribution "explodes" with the number of possible positions, velocities and actions. This is a well known issue of value iteration and therefore it is only applicable for very small problem-sets. For practically more relevant problems, there are a lot of other methods (quite a few of them are based on value iteration). More details and further references on value iteration can be found in the appendix A.2.

For the problems considered in this thesis, value iteration is still feasible within tolerable computation times. Furthermore, value iteration is simple to implement and provides insights on the transition distribution (which can be beneficial for debugging purposes). However, (by far) the largest part of the computational demand of the current implementation arises from using value iteration. As the selection of a planning algorithm is out of scope for this thesis and the use of value iteration leads to tolerable simulation run-times, no alternative has been implemented.

5.2.3 Number of mixture components

The number of Dirichlet mixture components directly determines the model's expressiveness. In the end, EM will use the model to *cluster* the given data-sets into sets with a similar transition-distribution (which of course is governed by the angle of the rotation-distortion). The number of these clusters is equal to the number of mixture components and it is not easy to intuitively specify a reasonable number. As the results show, and as one would naturally expect, if the number of mixture components is too small, strong averaging effects can be observed and the learned priors only reflect the structure of the task in a very coarse way (see the results in Sec. 4.1.2). But it is unclear if there will be strong overfitting effects for a too large number of clusters and "how many is too many"?

If the number of angles for the generated datasets is known, the problem becomes trivial. Especially at early stages of the experiments, it was desireable to have a small number of possible angles (say four) and all datasets are generated with one of these angles plus some small noise. In that case four mixture components will lead to very good solutions, whereas five or more components will lead to degenerate cases and less than four components will cause problems for some angles. At later stages of the experiments (where reasonable parameter settings had been found) it was much more desireable to have datasets with angles out of the whole range of zero to 360 degrees. Only that way, the structure extracting capabilities of the model could truly be investigated. However, in such a case it is no longer obvious how many mixture components would be required to extract the tasks' structure while at the same time avoiding strong overfitting effects.

Unfortunately, the EM algorithm makes the problem worse as too many mixture components will lead to degenerate cases and overlapping clusters. The simulation results have shown that (for the given problem parameters) a minimum number of about seven mixture components is required for a "good" performance, whereas the performance will no longer significantly increase when using more than 20 mixture components (but the computational demand does!). Therefore selecting the number of mixture components in this particular case is not too problematic. However this is a heuristic approach, instead of a general approach where the model would be able to automatically determine the required number of mixture components.

To avoid this problem, Maass, Neumann, Rückert have proposed to possibly use a Chinese Restaurant Process in order to determine the number of mixture components. Another solution would be the use of variational inference which is inherently capable of inferring the number of required mixture components from the data. But as already mentioned, the given problem setup is not too sensitive as far as the number of mixture components is concerned and the urgent necessity for automatically determining the number of mixture components was not given.

Another, less severe problem is that the model is not capable of interpolating between mixture components. By definition (through the introduction of the hidden **binary $\mathbf{z}$**-variables) a data-set has been created by a *single* mixture component. For the practical implementation this results in a slightly higher number of required mixture components and suboptimal results if the angle of a novel data-set lies exactly in-between the mean-angle of two cluster components. The use a continuous hidden variable would be nice from a conceptual point of view - however it is not straightforward and in this case the expected performance-increase is quite limited.

5.2.4 General aspects

On a first glance, the question whether the model is able to extract abstract knowledge, must be answered with a clear "yes". As the results show, the model is capable of extracting structural invariants and use this prior knowledge to rapidly adapt to novel but similar tasks. The model does not assume a rotation-distortion, yet it can nicely be used to compensate such a distortion. The question that remains is, whether the model in conjunction with the used methodology (EM) is able to truly "learn" to generalize?

Without question, the model is able to **store** knowledge on different layers of abstraction and with the use of EM it does **extract** abstract, structural knowledge from the given datasets. However, it only does so, due to a sophisticated design - it has been engineered to do exactly what it does. One could argue that the model simply reflects the engineer's assumptions on a certain class of problem tasks. Of course these assumptions are quite general; on an abstract level - yet the impression remains that the model has been "tailored" to be able to extract the tasks' structure. The implementation has also shown that there is quite some parameter-tuning as well as some heuristics necessary in order to be able to produce "good" results - which is undesirable for a system that should be able to easily generalize.

Generalization is inherently hard in a discrete setting. As already mentioned in 4.1.1 the (transition-) distributions used for learning simply require a large number of steps, otherwise they would be different from the true, underlying distributions. When using such flawed distributions for learning (i.e. fitting the model parameters) the results will be flawed as well because the model *learns from the wrong distributions.* Generalization is "easier" in a continuous setting since small changes in the arguments/parameters will usually result in small changes in the corresponding functions/distributions. Furthermore a discrete setting inherently includes *discontinuities* which are a severe problem for generalization.

A system that is truly capable of generalization would also need to be able to automatically set up a corresponding model as well as the topology of such a model (e.g. the number of latent variables, number of layers of abstraction, which layers get which (sensory) input, where does the output come from, ...). These processes are more related to topological structure learning (in terms of Bayesian models) and some research groups do not see a sharp distinction between topological structure learning and structural learning (in the sense of extracting structural invariants) - see [1] for examples and references to further work.

Consider the following example, taken from Gershman, Niv [1]: In a classical conditioning experiment, a rat receives an electric shock whenever a tone is presented. Soon, the rat will learn the causal relationship of tone and shock and will show reactions of fear when the tone is presented - see 5.1a. This stage of the experiment is called *acquisition* phase and it is already remarkable that the rat is able to quickly learn a causal relationship. In the second stage of the experiment, the *extinction* phase, the tone is presented without the shock and after quite a few observations, the rat will no longer show signs of fear when the tone is presented. In a naive interpretation, one could argue that the causal relationship has been *unlearned.* But with this explanation it is not possible to explain the third stage of the experiment, the *renewal* phase. If tone and shock are again presented simultaneously, the rat will only need very few observations to show the conditioned reactions again. It is therefore reasonable to suggest that the rat has actually learned that there is a latent variable that governs the stage of the task and that the simultaneity between tone and shock depends on this latent variable - see 5.1b. This interpretation is particularly interesting, because it claims that the rat was able to learn an internal model of the conditioning experiment including a latent (thus unobservable) variable. Learning such a model involved the "invention" of a latent variable, learning the (causal) connectivity structure as well as learning which sensory input is relevant and which is not. The approach used in this thesis is not capable of performing such learning tasks.

For a more complex conditioning-experiment with rats, see [30]. The paper presents results of a study where rats where conditioned under different contexts. Furthermore, the study shows that context-specificity of conditioning does only emerge in rats of a certain age, which suggests that the ability for structural learning might evolve with age.

Concluding the model used in this thesis, as well as similar models, might be just a way to express our current understanding of structural invariants in terms of a probabilistic model. In that case the capacity of expressing generalization would still be done by the engineer (with the model being one of his tools). But nonetheless, hierarchical Bayesian models could be one essential building block of a *generalizing* system or at least help in further understanding the

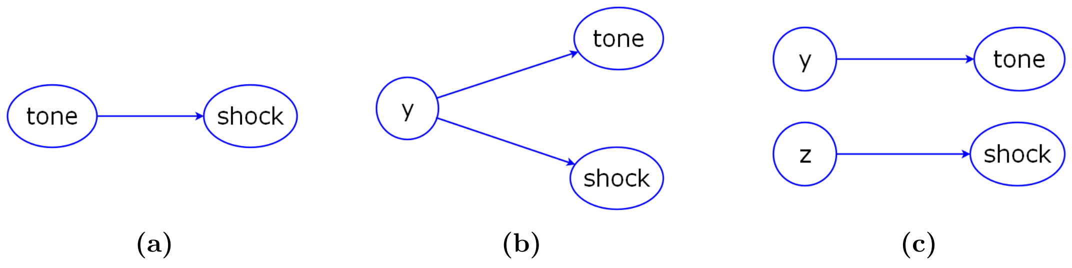


Figure 5.1: Possible structures for conditioning experiment - (a): (simple) causal relationship between the tone and shock. (b): tone and shock are governed by a latent variable y that corresponds to the stage of the experiment (acquisition, extinction or renewal). (c): tone and shock are each governed by independent random variables; in the given experiment these variables would be highly correlated, thus not independent. Concept for figures has been taken from [1] - where the full example is shown.

requirements and mechanisms of such a system. As far as technical applications are concerned, hierarchical Bayesian models used for structural learning seem to have a lot of very promising aspects to tackle cutting-edge problems in various domains.

5.3 Potential future research

The existing implementation has lead to promising results, that highlight the structural learning capabilities of the proposed model. However, the model is discrete and thus only allows for applications on discrete (or discretized) problems. This is a severe limitation - especially since the current implementation will not scale very well to large state- and action-spaces. On the other hand, the proposed approach is well-suited for applications on similar grid-world problems. Various kinds of movement-distortions, like shearings, scalings or translations, in such a grid-world can be compensated using the same approach as in this thesis - as long as the distortion shows some kind of structural invariant. Maass, Neumann, Rückert have also proposed a problem-task, where the grid-world contains obstacles with certain structural features (vertical versus horizontal obstacles for instance). Considering these problems, large parts of this thesis could potentially be used for such a problem-setting.

Generally speaking: any kind of problem-task where the state-transition distribution (of an agent in a grid-world) shows certain structural invariants, could be suitable for applying the methods presented in this thesis in order to *learn the tasks' structure*. When pursuing such goals it is reasonable and recommended by the author to work on the mentioned shortcomings (see previous Section 5.2). Especially the use of value iteration introduces quite large computational demands that could possibly be reduced significantly with alternative methods. But also the selection of the number of mixture components could have more impact on other problems and a system that is able to automatically infer the number of components from the observation data might be (very) desirable. It could therefore be interesting to implement a variational approach instead of using EM.

The approach taken in this thesis could provide inspiration on solving more complex problems. Imagine a wheeled robot, where one of the tires has a very low pressure, is damaged or perhaps the wheel got stuck. Depending on the wheel configuration (number and position of driven wheels versus passive wheels) this could easily lead to to a rotation-like movement distortion. Even more striking are applications with aerial agents: side-winds could be modelled as a kind of (translative) movement-distortion or a rudder that is not working properly will introduce a rotation-distortion on the corresponding axis. Anyway, for the application in a real-world (or close to real-world) scenario, a **continuous** model is required. Maass, Neumann, Rückert have already proposed such a model and one of the first steps could be the evaluation of this proposal on a more realistic problem-task.

As a long-term goal, potential research could be driven towards building truly generalizing systems and how hierarchical Bayesian models could play a role in such systems. From a neuroscientific perspective, biologically realistic models that are capable of structural learning are very interesting and might help in further understanding learning processes. Of course, both problems are topics of ongoing research and quite a number of groups are already working on these issues. So this should rather be seen as related research, rather than potential future research.

5.4 Concluding summary

Reviewing the initial problem of compensating a rotation-distortion on the movement of an agent that tries to reach a certain goal-position, the results are absolutely positive. The proposed model has proven to be able to extract the structure of the problem-task but also to provide transferable knowledge that can be used for adapting to a novel but similar instance of the task.

The main goal of this thesis was to demonstrate the model's capabilities of structure learning in terms of the given problem-task and provide a "proof-of-concept" implementation as well as a simulation environment for evaluating the qualities of the model. This goal has been reached with the implementation on one hand and with the results of the performed experiments in the simulation environment on the other hand. The results show that the agent is able to *rapidly* adapt to a new task by using the prior-knowledge provided by the hierarchical Bayesian model. Further results compare this capability against a system that has no prior-knowledge and, as expected, such a system performs worse for the given problem-setting. It was also possible to gain some intuitive insight on how the model captures the structure of the task as well as the evolution of the anticipated structure of a new task over the first time-steps. Finally the results also highlight a few details of the process of fitting the model to given observations (using EM) and the choice of the model parameters, such as the number of mixture components.

In the course of this thesis, some shortcomings of the approach have emerged (number of required mixture components can not be inferred from the data, EM leads to local maxima and potentially to degenerate cases). The implementation has a few known flaws that are either tolerable or out of scope for this thesis (computational demand of value iteration, heuristics in EM and value iteration). These issues could be the main-part of immediate future work. However, the expected improvements have to be compared against the limits of the model and it might be more beneficial for more realistic problems to use a different model that is capable of handling *continuous* scenarios. In the long run, such a continuous model seems more promising than a model that is inherently restricted to discrete problem-tasks.

Structural learning promises to solve some of the cutting-edge problems in machine learning or at least provide some inspiration for novel methods. Furthermore, it might help in understanding neurological learning processes. This thesis illustrates the potential benefits on a simple, abstract task - especially the aspect of *fast learning*, i.e. the capability to rapidly adapt to a novel but similar task using abstract prior-knowledge that has been gained from a number of previous observations.

A Appendix

A.1 Details on M-step of basic model

Continuing from Equation (3.15), the gradient with respect to $\alpha_{k,nl}$ needs to be computed:

$$\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \left(\sum_{j=1}^{N} q_j(k) \sum_{n=1}^{|A|} \ln W(\boldsymbol{\alpha}_{k,n} + \mathbf{m}_{j,n}) - \ln W(\boldsymbol{\alpha}_{k,n}) \right), \tag{A.1}$$

with the Dirichlet weighting function defined as (see also Sec. 3.2.2):

$$W(\boldsymbol{\alpha}) := \frac{\mathit{\Gamma}(\alpha_1) \cdots \mathit{\Gamma}(\alpha_K)}{\mathit{\Gamma}(\alpha_0)}, \tag{A.2}$$

and

$$\alpha_0 = \sum_{k=1}^{K} \alpha_k \tag{A.3}$$

Using the logarithmic-gamma function $\mathit{\Gamma}_{\ln}(x) := \ln(\mathit{\Gamma}(x))$, Eq. (A.1) can be rewritten as:

$$\begin{aligned} \sum_{j=1}^{N} q_j(k) \frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \sum_{n=1}^{|A|} [\mathit{\Gamma}_{ln}(\alpha_{k,n1} + m_{j,n1}) + \cdots + \mathit{\Gamma}_{ln}(\alpha_{k,nL} + m_{j,nL}) - \mathit{\Gamma}_{ln}(\alpha_{0k,n} + R_{j,n})] - \\ -[\mathit{\Gamma}_{ln}(\alpha_{k,n1}) - \cdots - \mathit{\Gamma}_{ln}(\alpha_{k,nL}) + \mathit{\Gamma}_{ln}(\alpha_{0k,n})], \end{aligned} \tag{A.4}$$

with $L := |\mathit{\Delta} P|$ and $R_{j,n}$ defined as in Eq. (3.17). Since the optimization only considers a single mixture component at a time (because it can be optimized separately), the derivative can be “pulled into” the first sum.

Notice the difference between the indices of the (partial) derivative in non-italics and the indices in italics. Further notice that the derivative simply vanishes if the indices (in italics and non-italics) are different - only if they are exactly equal, the following derivative remains:

$$\nabla \alpha_{\mathrm{k,nl}} = \sum_{j=1}^{N} q_j(k) \left(\psi(\alpha_{\mathrm{k,nl}} + m_{j,\mathrm{nl}}) - \psi(\alpha_{\mathrm{k,nl}}) + \psi(\alpha_{\mathrm{0k,n}}) - \psi(\alpha_{\mathrm{0k,n}} + R_{j,\mathrm{n}}) \right), \tag{A.5}$$

with psi being the digamma function which is the first-order derivative of the logarithmic gamma function: $\psi(x) := \frac{d}{dx} \mathit{\Gamma}_{\ln}(x)$.

A.2 Value iteration

The method of *value iteration* is, as the name suggests, based on the evaluation of value functions, such as most reinforcement learning algorithms and even a wide range of other optimization algorithms. Richard E. Bellman has contributed substantial groundwork for the field of reinforcement learning - especially with the introduction of the *Bellman optimality equation* and the method(s) of *dynamic programming* to solve such equations [31]. Value iteration also arises from this work. A more recent discussion can be found in *the* standard textbook on reinforcement learning by Sutton, Barto [32] (which also contains introductory chapters and some background on dynamic programming and value iteration). The following introduction is based on the textbook by Sutton, Barto as well as the survey by Kaelbling, Littman, Moore [33].

A.2.1 Basics

There are basically two kinds of value functions:

State-value function: In very simple terms, this function provides a measure on "how good" it is for the agent to be in a certain state. It does so by giving the expected *return* of a state - which is the expected (cumulative/discounted) future reward. Approximations to the state-value function are commonly used instead of a complete model of the agent's environment (prediction problems).

Action-value function: The action-value function also gives a measure on the expected return of a certain state but with respect to a particular action for the immediate next step. It is therefore commonly used for planning (control problems), because the action which has the highest action-value for a certain state is the "best" action to take. In fact, the state-value function can be derived from the action-value function by simply taking the maximum action-value (of all actions) over every state.

The state-value function is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | s_t = s\right] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right], \tag{A.6}$$

where π denotes a policy $\pi(s,a)$ which is used for action-selection. γ is a discount factor which has the effect (if it is smaller than one) that the significance of rewards decays over time. t denotes the current time-step, R stands for the return (cumulative discounted reward, r_t is the reward at time t and s_t represents the state at time t.

The action-value function has a similar definition:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_t | s_t = s, a_t = a\right], \tag{A.7}$$

with a or a_t denoting a specific action. The action-value function differs from the state-value function only by the specification of a particular action for the *immediate* next step and then following the policy π. By considering all possible actions and choosing the one with the highest expected return, the state-value for the current state can be found:

$$V(s) := \max_a Q(s,a) \tag{A.8}$$

With $V^*(s)$, denoting the optimal value-function (under an optimal policy π^*) the Bellman equation leads to the following:

$$V^*(s) = \max_a Q^{\pi^*}(s,a) = \max_a \mathbb{E}\left[r_{t+1} + \gamma V^*(s_{t+1}|s_t = s, a_t = a)\right] \tag{A.9}$$

This means that the expected Return for the current state is computed by taking the maximum over all actions and for each action adding the corresponding (immediate) reward to the expected future return (cumulative discounted reward). Value iteration is one way to solve this recursive equation in a dynamic-programming fashion. It was derived by transforming the equation from above into an iterative update rule. The value-iteration algorithm has the following form:

```
Initialize $V(s)$ arbitrarily
**repeat**
  **for all** $s \in S$ **do**
    **for all** $a \in A$ **do**
      $Q(s,a) := R(s,a) + \gamma \sum_{s' \in S} T(s,a,s')V(s')$
    **end for**
    $V(s) := \max_a Q(s,a)$
  **end for**
**until** policy good enough
```

algorithm from [33].

$T(s,a,s')$ is the state-transition probability and it specifies the probability of ending up in state s' when currently being in state s and executing action a. This is usually understood as the *full model of the agent's environment*, since value iteration needs the full state-transition probability for all possible combinations of s,a and s'. In a lot of realistic applications, the computation of all these transition-probabilities is not possible or at least computationally demanding - it is usually more desirable to have a planning algorithm that is able to deal with an incomplete model of the environment or one that does not need such a model at all (see also [32]). For the problems regarded in this thesis it is possible to compute the desired quantities, however they introduce a non-neglectable computational-cost.

From the pseudo-code given above, it is easy to see that the computational demands of value-iteration *explode* with an increasing size of the state-set S or the action-set A. Besides the full model of the environment, this is the most severe problem with value iteration - for problem tasks of practical interest it is almost never possible to run value iteration within tolerable runtimes - in most real-world problems it is utterly impossible; especially when real-time planning is required. The problem tasks, handled in this thesis are already close to the border of tolerable simulation runtimes and the largest part of the computational effort is necessary for the value-iteration.

The great advantage of value iteration is that it has solid theoretical foundations as well as convergence proofs (it is guaranteed to converge towards an optimal value-function which can be used to derive an optimal policy - however this convergence is only guaranteed when approaching infinity). The pseudo-code above states that the algorithm is iterated until the policy is “good enough”. In a practical implementation this is usually interpreted the following way: if the state-value function does no longer significantly change between two iterations (i.e. some error-measure is below a certain ϵ), stop iterating. This is a somewhat crude approach, but it turned out to be effective and sufficient for a lot of applications.

Many algorithms (in reinforcement learning) are based on the idea of value iteration but they perform the same computations or approximations with less computational efforts - see Sutton, Barto [32] for more on this.

A.2.2 Non-deterministic Action selection

Assuming that the action-value function has been computed - it can easily be used for planning by simply taking the action that has the highest value for the current state. If the value function does not change, this process is deterministic. In some scenarios it might be beneficial to have a non-deterministic action selection mechanism. There are a quite a few methods in reinforcement learning that do not have a full model of the environment and thus rely on a certain degree of *exploration* - i.e. the agent has to “try” suboptimal actions because their current action-value is just an estimate and if this estimate is based on very few samples, it might turn out that the supposedly suboptimal action is better than expected (the problem is that the reward will usually occur many steps after taking the action and somehow has to be assigned to all actions that led towards it, which makes it hard to easily decide on the quality of a certain action).

Even for the case of value iteration (such as in this thesis) there are special cases that could lead to certain problems. Imagine that the model has seen datasets with four different rotations (0°, 90°, 180°, 270°) and that the agent's initial position is already close to the goal-position. Unless the agent performs a step, it can not infer the rotation-angle of the new episode - the planning algorithm (value-iteration) will then probably propose an acceleration of (0,0) because the risk to move away from the goal position, thus receive a negative reward, is quite high. However, the action (0,0) will not help determining the rotation-angle of the current episode, since it does not (and cannot) contain any rotation-information. In fact, this particular action will look very similar at all prior-mixture-components. All it would contain, is information on the variance σ^2 (see (3.25)). The agent would then simply remain at the initial position and always choose action (0,0), because it has the highest action-value. In such a case it makes sense to select a suboptimal action! There are two common ways to implement such an action-selection behavior:

Softmax action-selection: Selects an action according to the expected reward based on a temperature parameter. For high temperatures, all actions have nearly the same probability. For lower temperatures, the action selection is based more and more on the expected reward (i.e. two actions with an almost equal expected reward will still be selected with almost equal probability). Unfortunately, finding good values for the temperature parameter is not trivial. The advantage of softmax is that selecting the second-best action is much more likely than, for example, the worst action.

ϵ-greedy action-selection: Actions are usually selected according to their expected reward but with a small probability ϵ a random action is selected. It is generally easier and more intuitive to choose a good ϵ-value. In difference to softmax, the random action selection is not affected by the expected reward, which can be problematic if there is a large number of quite "bad" actions.

A.3 Details on M-step of extended model

The goal of the M-step is the maximization of the expected value of the log-likelihood (of the latent variable) - in this case with respect to the parameters of the Dirichlet-mixture ($\boldsymbol{\alpha}$). In this case, each of the K mixture-components can be optimized separately. See Eq. (3.23) or the corresponding Section 3.3.2 for more information.

The log-likelihood for the basic model has a very similar form - the results for the corresponding M-step are still valid. For the extended model, there is an additional hyper-prior term for every mixture component. This term needs to be taken into account as well when optimizing with respect to $\boldsymbol{\alpha}$. The following derivation shows the steps to compute the gradient of $P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})$ with respect to $\boldsymbol{\alpha}_k$. For the full M-step, these results need to be added to the results of the basic M-step (see Eq. (3.23)).

Taking the ln of the prior distribution $P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa})$, given by equation (3.20), we get:

$$\ln P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa}) = \sum_{n=1}^{|A|}\sum_{l=1}^{|\Delta P|} \ln\left(\sum_{b=1}^{B} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b)P(b|\mathbf{x})\right) + \ln\left(\mathcal{N}(\alpha_{0k,n}|\boldsymbol{\gamma})\right) \tag{A.10}$$

The gradient with respect to $\boldsymbol{\alpha}_{\mathrm{k}}$ is:

$$\begin{aligned}\frac{\partial}{\partial \boldsymbol{\alpha}_{\mathrm{k}}} \ln P(\boldsymbol{\alpha}_k|\boldsymbol{\kappa}) = &\sum_{n=1}^{|A|}\sum_{l=1}^{|\Delta P|} \frac{1}{\sum_{b=1}^{B} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b)P(b|\mathbf{x})} \cdot \\ &\cdot \underbrace{\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}}\left(\sum_{b=1}^{B} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b)P(b|\mathbf{x})\right)}_{A)} + \underbrace{\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \ln\left(\mathcal{N}(\alpha_{0k,n}|\boldsymbol{\gamma})\right)}_{B)}\end{aligned} \tag{A.11}$$

Notice that the indices of the derivative k,n,l (normal text) are not the same as the indices of the sums k,n,l (in italics). They share the same range of values and, of course, the derivative needs to be evaluated for every possible triple.

Still missing are the "inner derivations" A) and B). The second term - B) - is simply the derivative of the ln of a Gaussian. Notice that $\alpha_{0k,n}$ is the sum over all $\alpha_{k,nl}$ (see Eq. (3.18)), thus the derivative will vanish for all, except one term (given by the non-italics indices).

$$\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \ln\left(\mathcal{N}(\alpha_{0k,n}|\boldsymbol{\gamma})\right) = -\frac{1}{\sigma_0^2}\left(\alpha_{\mathrm{k,nl}} - \mu_0\right), \tag{A.12}$$

with the parameter-vector $\boldsymbol{\gamma} = \langle \mu_0, \sigma_0 \rangle$.

The derivative for A) has the following form:

$$\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \left(\sum_{b=1}^{B} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b) P(b|\mathbf{x}) \right) = \sum_{b=1}^{B} \left(\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \mathrm{Beta}(\mu_{k,nl}|\boldsymbol{\lambda}_b) \frac{\partial \mu_{k,nl}}{\partial \alpha_{\mathrm{k,nl}}} \right) P(b|\mathbf{x}), \tag{A.13}$$

where the derivative of the Beta-distribution is again a Beta-distribution with a new normalization factor

$$\frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \mathrm{Beta}(\mu_{k,nl}|\underbrace{c_b, d_b}_{\boldsymbol{\lambda}_b}) = \left(\frac{c_b - 1}{\mu_{k,nl}} - \frac{d_b - 1}{1 - \mu_{k,nl}} \right) \mathrm{Beta}(\mu_{k,nl}|c_b, d_b) \frac{\partial \mu_{k,nl}}{\partial \alpha_{\mathrm{k,nl}}} \tag{A.14}$$

using the parameter-vector $\boldsymbol{\lambda}_i = \langle c_i, d_i \rangle$ where c_i,d_i are the parameters of a single Beta-distribution.

The last missing part is the "inner, inner derivation"

$$\frac{\partial \mu_{k,nl}}{\partial \alpha_{\mathrm{k,nl}}} = \frac{\partial}{\partial \alpha_{\mathrm{k,nl}}} \frac{\alpha_{k,nl}}{\alpha_{0k,n}} = \begin{cases} \frac{1}{\alpha_{\mathrm{0k,n}}} - \frac{\alpha_{\mathrm{k,nl}}}{\alpha_{\mathrm{0k,n}}} & \text{if k=}k\text{, n=}n\text{, l=}l \\ -\frac{\alpha_{\mathrm{k,nl}}}{\alpha_{\mathrm{0k,n}}} & \text{otherwise} \end{cases} \tag{A.15}$$

If the italics indices do not equal the normal indices, the element in the numerator can be regarded as a constant and the derivative results from the term in the denominator $\alpha_{0k,n}$ (the sum over all $\alpha_{k,nl}$ will also contain $\alpha_{\mathrm{k,nl}}$). However if all indices are equal, the derivation-variable appears both in the numerator as well as in the denominator.

Bibliography

[1] S. J. Gershman and Y. Niv, “Learning latent structure: carving nature at its joints.,” *Current Opinion in Neurobiology*, vol. 20, no. 2, pp. 251–256, 2010.

[2] D. A. Braun, A. Aertsen, D. M. Wolpert, and C. Mehring, “Motor task variation induces structural learning,” *Current Biology*, vol. 19, no. 4, pp. 352–357, 2009.

[3] D. A. Braun, C. Mehring, and D. M. Wolpert, “Structure learning in action.,” *Behavioural Brain Research*, vol. 206, no. 2, pp. 157–165, 2010.

[4] K. J. Astrom and B. Wittenmark, *Adaptive Control.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 1994.

[5] C. Kemp, A. Perfors, and J. B. Tenenbaum, “Learning domain structures,” *Proceedings of the 26th annual conference of the cognitive science society*, p. 720–725, 2004.

[6] C. Kemp and J. B. Tenenbaum, “The discovery of structural form,” *Proceedings of the National Academy of Sciences*, 2008.

[7] D. A. Braun, S. Waldert, A. Aertsen, D. M. Wolpert, and C. Mehring, “Structure learning in a sensorimotor association task,” *PLoS ONE*, vol. 5, no. 1, p. 8, 2010.

[8] H. F. Harlow, “The formation of learning sets.,” *Psychological Review*, vol. 56, no. 1, pp. 51–65, 1949.

[9] Z. Reznikova, *Animal intelligence. From individual to social cognition*, vol. 37. Cambridge University Press, 2007.

[10] C. Kemp and J. Tenenbaum, “Structured statistical models of inductive reasoning,” *Psychological Review*, vol. 116, no. 1, pp. 20 –58, 2009.

[11] E. J. A. Turnham, D. A. Braun, and D. M. Wolpert, “Inferring visuomotor priors for sensorimotor learning,” *PLoS Computational Biology*, vol. 7, no. 3, p. 13, 2011.

[12] D. E. Acuna and P. Schrater, “Structure learning in human sequential decision-making,” *PLoS Computational Biology*, vol. 6, no. 12, p. 12, 2010.

[13] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4. Springer, 2006.

[15] D. Heckerman, “A tutorial on learning with bayesian networks,” *Innovations in Bayesian Networks*, vol. 1995, no. November, p. 33–82, 1996.

[16] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[17] C. Huang and A. Darwiche, “Inference in belief networks: A procedural guide,” *International Journal of Approximate Reasoning*, vol. 15, pp. 225–263, 1996.

[18] D. J. C. MacKay, “Introduction to Monte Carlo methods,” in *Learning in Graphical Models* (M. I. Jordan, ed.), NATO Science Series, pp. 175–204, Kluwer, 1998.

[19] W. L. Buntine, “Operations for learning with graphical models,” *Journal of Artificial Intelligence Research*, vol. 2, pp. 159–225, Dec. 1994.

[20] M. I. Jordan, “An introduction to variational methods for graphical models,” in *Machine Learning*, pp. 183–233, MIT Press, 1999.

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society Series B Methodological*, vol. 39, no. 1, pp. 1–38, 1977.

[22] V. K. Mansinghka, C. Kemp, J. B. Tenenbaum, and T. L. Griffiths, “Structured priors for structure learning,” *Methods*, vol. 136, no. 11, pp. 3802–3821, 2006.

[23] N. Friedman, *The Bayesian Structural EM Algorithm*, vol. 98, pp. 129–138. Citeseer, 1998.

[24] J. Pena, “An improved bayesian structural em algorithm for learning bayesian networks for clustering,” *Pattern Recognition Letters*, vol. 21, no. 8, pp. 779–786, 2000.

[25] G. Peng and L. Naixiang, *An EM-MCMC algorithm for Bayesian structure learning*. IEEE, 2009.

[26] C. Kemp, A. Perfors, and J. B. Tenenbaum, “Learning overhypotheses with hierarchical bayesian models,” *Developmental Science*, vol. 10, no. 3, pp. 307–321, 2007.

[27] S. Borman, “The expectation maximization algorithm a short tutorial,” Tech. Rep. x, 2009.

[28] G. McLachlan and D. Peel, *Finite Mixture Models*, vol. 44. Wiley-Interscience, 2000.

[29] G. J. McLachlan and T. Krishnan, *The EM Algorithm and Extensions*, vol. 274. Wiley, 1997.

[30] C. S. L. Yap and R. Richardson, “Extinction in the developing rat: an examination of renewal effects.,” *Developmental Psychobiology*, vol. 49, no. 6, pp. 565–575, 2007.

[31] R. Bellman, *Dynamic Programming*, vol. 11. Princeton University Press, 1957.

[32] R. S. Sutton and A. G. Barto, “Reinforcement learning,” *Learning*, vol. 9, no. 1, pp. 1–23, 1998.

[33] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, no. 1, pp. 237–285, 1996.