AUTONOMOUS LEARNING OF OBJECT BEHAVIOR CONCEPTS AND MODELS THROUGH ROBOTIC INTERACTION

Thesis for obtaining the title of
Doctor of Engineering
of the Faculty of Mathematics and Computer Science
of Saarland University

by
Sergio Roa Ovalle, M.Sc.

Saarbrücken
February 2016

Day of Colloquium	November 16th 2016
Dean of the Faculty	Univ.-Prof. Dr. Frank-Olaf Schreyer
Chair of the Committee	Prof. Dr. Antonio Krüger
Reporters	
First reviewer (Supervisor):	Prof. Dr. Hans Uszkoreit
Second reviewer	Prof. Dr. Oliver Brock
	(Technische Universität Berlin)
Academic Assistant	Dr.-Ing. Ivana Kruijff-Korbayova

Dedicated to my family,

especially to my parents, María Elvia y José.

ABSTRACT

We need robots that learn from the environment by interacting with it, and deduce models of causal relations and associations from these interactions. In this dissertation, we address the particular problem of predicting object trajectories when manipulating objects, using robot arm pushes. To solve this problem we derive models which can describe the behavior of objects put in motion. For a learning robot it is essential to learn in an incremental and active way when new information is coming in, and to do so without losing generalization and without overfitting. First, we tackle this problem by estimating the density of a sensorimotor space after a robot performs a new action by using a modification of the incremental Growing Neural Gas (RobustGNG) algorithm. RobustGNG performs a space quantization which is robust to noise and overfitting issues. Subsequently, we infer models useful for prediction of object trajectories in terms of object poses. The same machinery is useful for obtaining more coarse-grained predictions, for instance categorizations of object behaviors. Last but not least, these prediction models provide a qualitative and temporal description of the state space, so that they can eventually be used in planning tasks. We infer cause-effect models by using RobustGNG's results in a new version of the CrySSMEx algorithm to generate substochastic finite-state machines.

ZUSAMMENFASSUNG

Wir brauchen Roboter, die durch Interaktion mit der Umwelt von dieser lernen und aus dieser Interaktion Modelle für Kausalrelationen und -zusammenhänge ableiten können. In dieser Dissertation wird das spezielle Problem der Vorhersage von Objektbewegungen durch Manipulation von Objekten durch Roboterarme behandelt. Um dieses Problem zu lösen, werden Modelle hergeleitet, die das Verhalten von in Bewegung versetzten Objekten beschreiben können. Für einen lernenden Roboter ist es essentiell wichtig, schrittweise und aktiv zu lernen wenn neue Informationen kommen, ohne dabei die Generalisierung zu verlieren oder Überanpassung zu generieren. Das Problem wird zunächst dadurch angegangen, dass die Dichte eines sensomotorischen Raumes geschätzt wird, nachdem ein Roboter eine neue Bewegung ausgeführt hat. Hierfür wird eine Abwandlung des inkrementellen Growing Neural Gas (RobustGNG) Algorithmus verwendet. Die durch RobustGNG vorgenommene Quantisierung des Raumes ist unanfällig für Störungen und Überanpassung. Daraufhin können Modelle abgeleitet werden, die für die Vorhersage von Objektbewegungen bezüglich ihrer Objektstellung nützlich sind. Derselbe Mechanismus ist nützlich, wenn gröbere Vorhersagen getroffen werden sollen, z.B. bei der Kategorisierung von Objektverhalten. Zu guter Letzt bieten diese Vorhersagemodelle eine qualitative und zeitliche Beschreibung des Zustandsraumes, sodass sie schließlich auch bei Planungsaufgaben verwendet werden können. Es werden Ursache-Wirkungs-Modelle abgeleitet, indem die Ergebnisse des Robust GNG in einer neuen Version des CrySSMEx-Algorithmus verwendet werden, um substochastische Zustandsmaschinen zu erzeugen.

Imagination is more important than knowledge.

Ich habe keine besondere Begabung, sondern bin nur leidenschaftlich neugierig.

— Albert Einstein

Science may be described as the art of systematic over-simplification – the art of discerning what we may with advantage omit.

— Karl Popper

Can machines think?

— Alan Turing

ACKNOWLEDGMENTS

I am especially indebted to my advisor Dr. Geert-Jan Kruijff, who gave me the opportunity to work in the collaborative EU-funded project CogX. I also wish to express my gratitude to Henrik Jacobsson who introduced me to the area of cognitive robotics. I am particularly grateful to former CogX participants, especially Marek Kopicki and Thomas Mörwald. Without their contributions to the project this dissertation would not have been possible. Many thanks to all CogX partners, with whom we had interesting dialogues around the field.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 LEARNING MODELS FOR MANIPULATION. STATE OF THE ART.

Robots need to obtain abilities for manipulating objects, if we want them to carry out complex tasks which are easily accomplished by humans. They need to be able to solve tasks by identifying physical object properties which are useful in some context. For instance, a robot might want to move a box by sliding it in order to grasp it properly and then serve its content in a bowl, as exemplified in Fig. 1. Recognizing these properties implies that a robot should be able to predict the consequences of certain actions applied to different objects in the environment. Predicting is a consequence of previous knowledge acquisition. One possible approach is the encoding of knowledge by means of predefined rules, which is tedious work given the variety and complexity of features for objects of any kind that we find in the world. Moreover, it is impossible to do this robustly, flexibly and scalably. Therefore, the approach of learning by interacting with the environment is an evident solution.

We consider here the problem of predicting consequences by applying a restricted set of actions to simple geometrical objects. We have developed novel machine learning algorithms (Robust Growing Neural Gas - RobustGNG and improved Crystallizing Substochastic Sequential Machine Extractor - CrySSMEx), which are able to accurately predict an object pose, given some motor action performed by the robot arm on the object and previous object poses. Prediction is in turn essential for a robot to plan and execute actions in order to achieve some desired goal in the context of solving some task. For example, a robot might predict that if it pushes an object forward and then laterally, the object will then be located in an intended position for it to grasp.

In this dissertation, the prediction models are constructed in such a way that they can be further applied for these planning or control tasks, because they can infer a set of rules or causal relationships among different features in the environment across time. Markov Decision Processes (MDPs) are a well established planning framework which suits our models. However, we only test the prediction ability of these machines, and leave the evaluation of planning for future research.

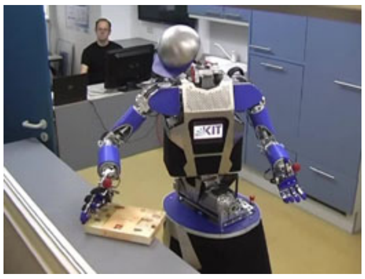


Figure 1: The robot pushes an object in order to grasp it from the edge of the sideboard. Photo: Humanoids and Intelligence Systems Lab/Karlsruhe Institute for Technology [46]

The work presented here is inspired by the fact that humans and animals in general are able to predict and learn from a dynamic environment. Theories of cognitive development like the theory of affordances [17] attempt to explain how creatures are able to acquire sensorimotor skills when they are faced with the different features found in the environment. For instance, surfaces afford posture, locomotion, collision, manipulation, and in general behavior [17]. Moreover, affordances of objects refer to the perceived and actual properties of objects which determine how things could possibly be used [44]. It follows from the above facts that intelligent creatures should then properly predict the consequences of actions on a surface given their own body configuration. After this has been accomplished, they need to be able to plan actions on the basis of newly gathered experience. For our purposes, we consider here a special case of affordance learning in robots, namely that of predicting consequences of pushing simple geometrical objects. Pushing is a useful type of interaction to consider, as already exemplified in Fig. 1. Moreover, it can be helpful for identifying dynamical properties of objects in an unknown environment.

Exploring the surrounding environment in an efficient way is a key issue for dealing with computational complexity issues. Robots are faced with many features in the environment, which they will have to process and compress in an efficient way. Taking into account the problem we are tackling in this dissertation, we propose several new methods which aim to reduce the computational complexity with respect to the selection of actions required for learning about pushing actions. For this purpose, we argue that we need robots that extend their knowledge in an incremental, online and active fashion. In robotic contexts, applying these learning approaches is challenging, because one needs to address issues like robustness to noisy environments, overtraining and high dimensionality.

During and/or after these exploratory phases, it is essential that a robot with manipulation skills induces models which could be useful for carrying out subsequent manipulation tasks. Our approach addresses this issue by building models or methods useful for planning, after the exploratory phase.

Online and active learning strategies have been applied and studied recently in robotic problems. Online and active density estimation methods, e.g. based on kernel methods, have been studied in the context of discriminative models for object classification [34, 59]. In robotic manipulation, exploratory and active learning have been applied for the online acquisition of grasping skills [32, 35]. For instance, in [32] an active reinforcement learning approach is applied to enrich object representations and object-specific grasp affordances. In [35], an exploratory developmental learning method is used to first construct object models and then grasp densities.

Although more focused on aspects more relevant for socially guided learning, the work in [68] looks into exploratory behavior for robots with social skills but limited manipulation skills. They consider socially guided exploration for solving simple tasks. In this setting (see Fig. 2), the authors consider self-motivated exploration of a limited set of possible actions, in a similar way as in [47]. The robot learns the names of three buttons and is motivated by a tutor to turn them all on. Self-motivated exploration can have different motivation

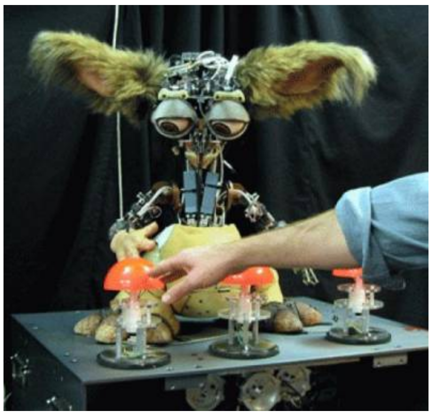


Figure 2: Leonardo robot in a learning scenario with three buttons and a human tutor [68].

sources. In [68], an activity drive, a novelty drive and a mastery drive are self-motivated exploratory drives. While performing exploratory actions, the robot also performs actions aiming to solve the current task. For these purposes, reinforcement learning strategies are used, specifically task option policies. The main drawback of the work is that the set of states is predefined for the task, so that the objective is to learn the policy for the task given a set of states and actions. In contrast, in our work the learning drive of the robot can be defined as a mastery and self-motivated drive, because the robot selects actions with the purpose of reducing the prediction error. Furthermore, our work presents an approach for robot learning where the states and actions are not explicitly given a priori, in a similar way as in [42]. In this simulated scenario (Fig. 3), the robot can grasp a block if the hand and block are colliding. In addition, the agent can observe two floating objects which it cannot interact with, to test the method with distractor objects. The authors designed a system which can discover qualitative states and actions, based on motor, magnitude and direction of change variables. After this discretising step, rules (models of causal relationships) are learned by using Dynamic Bayesian Networks (DBNs). The robot learns to move the block, push the block to the floor and pick up the block. The DBNs principles are also similar to the probabilistic models we apply in our methods, where dynamic state/action pairs are the building blocks of these prediction models.

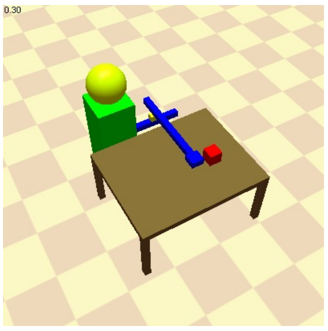


Figure 3: Simulated robotic scenario for qualitative learning [42]

The problem of predicting object trajectories after a rigid body pushes it has already been addressed in previous work. In [30, 29], the authors applied offline probabilistic models for density estimation of object poses, specifically Gaussian kernel density methods. A completely different approach with no learning phase was adopted in [31], where the physics principle of mimimum energy was used. In contrast, our work aims to apply incremental, active and online techniques for learning. We also focus on learning models which are potentially useful for planning, as stated above. For that purpose, we apply the CrySSMEx algorithm [25] for extracting rules about causal relationships of action/outcome pairs. In [58], the authors present a learning system which is conceptually similar to our approach. A discrete set of actions is predefined and a probability model of displacement is updated after the robot performs an action. Then, a task space planner based on Rapidly-exploring Random Trees is used to solve different manipulation tasks.

The problem we consider here is closely related to previous work on affordance learning [62, 47, 72, 51, 3, 32, 35, 24], among others. In [72], the robotic arms perform some random actions such as pushing or lifting which are mapped to visual features during a learning phase. Then, by implementing a planning framework, these affordance relations can be used for solving different tasks, for instance bringing an object to a target position. [24] describes a system specialized for pushing and pulling actions for positioning objects, which can learn and control these behaviours at each time step.

In [47], the robots are more autonomous and enter into different developmental stages on the basis of a learning progress measure, which enables the robot to decide what action to perform next in the presence of different objects, emulating a curious agent. The authors demonstrate that the agent "discovers" affordances while it starts to perform certain actions repeatedly. This approach is related to active learning, in the sense that an agent needs to choose a learning example so that the information gain is maximized, thus minimizing the amount of examples needed. From now on, we will simply call these learning strategies *active learning*.

In [51], the objective was to extract visual features from random pushing actions which can be used for obtaining categorizations of affordances via unsupervised learning. In the field of cognitive robotics, a related theory has also been developed in recent years, called *object-action complexes* (OACs) [36]. OACs aim at providing a universal representation enabling efficient planning and execution of actions in different levels of a cognitive architecture [36]. OACs combine search capabilities based on theorem-proving rules and the object and situation oriented concept of affordance, plus the logical framework of the situation calculus for planning [36].

Affordances are also strongly related to the emergence of concepts [3]. A new behavior observed in an object or a new way to adapt the body and apply it to an object to achieve some goal might lead to new linguistic descriptions of these situations. As we mentioned above, we use probabilistic methods to discriminate among different object behaviors given some actions.

The learning processes presented above are elements of theories of Developmental or Cognitive Robotics. Different learning stages are necessary for a robot to accomplish a certain goal. Motor skills influence the ability of a robot to express its knowledge about an action and, at the same time, language through communication can influence the way motor actions are performed. We focus our investigation in the central role that plays self-motivated exploration in this developmental learning chain. This is an essential precondition for further skills like verbal communication and the achievement of common goals when robots and humans are interacting. For this purpose, the robots need to acquire models which are useful for planning and concept acquisition that can be incorporated in an interactive setting.

Developmental Robotics is a newly established field. "The basic research assumption of this field is that true intelligence in natural and (possibly) artificial systems presupposes three crucial properties: *embodiment* of the system, *situatedness* in a physical or social environment, and a prolonged *epigenetic developmental process* through which increasingly more complex cognitive structures emerge in the system as a result of interactions with the physical or social environment" [63]. Developmental Robotics, according to [63], emerged partly as a need for having robots with close to human levels of intelligence. The robotic architectures did not scale up due to the amount of pre-programming and knowledge engineering that are needed for addressing the complexity of robotic tasks. Developmental approaches are inspired by developmental psychology [47]. In [47], two characteristics of child development inspire the application of resembling techniques for robots:

- Development is progressive and incremental. "Children undergo a developmental sequence during which a new skill is only acquired when the associated cognitive and morphological structures are ready." [47].

- Developmental learning presupposes it is autonomous and active. Children decide what tasks are more interesting, even when parents help by scaffolding the infant environment.

1.2 CONTRIBUTIONS OF THIS DISSERTATION

We set up learning robotic scenarios involving a robotic arm and a geometrical object to address the problem of predicting an object pose, given a motor action performed on an aspect of the object, and previous object poses. To solve this problem, we apply learning methods which act upon features from vision, simulation and manipulation systems. These features are basically object poses, including finger effector poses, which together define sensorimotor spaces. By performing pushing actions on the object, the robot will be able to construct prediction models from these interactions by means of learning algorithms. In order to discretize the feature spaces, we tackle the problem of density estimation (quantization) of sensorimotor spaces in an interactive environment. Additionally, we tested the classification ability of these prediction models to discriminate among a set of abstract object behaviors.

In this dissertation, we improved and tested new methods for dealing with these learning problems. They are briefly summarized in the following list:

- We developed incremental and active quantization methods for density estimation of sensorimotor spaces applied to robot learning scenarios. In this way, following our reasoning, we aim to present new strategies leading to online learning and efficient ways for selecting actions. For these purposes, we designed and applied an improved version of the incremental learning algorithm Growing Neural Gas (RobustGNG).

- We developed and improved the CrySSMEx method to perform an automatic construction of models which exhibit the dynamics of robot/object interactions as probabilistic finite state machines. These models are useful for planning, but a proof of this is still future work.

- We prove that these probabilistic finite state models are useful for classification of more abstract behavior patterns, i.e., for concept learning.

These three aspects of learning rely on the assumption that we can derive state space models by using information-theoretic mechanisms to reduce the dimensionality of the sensorimotor state space. On the one hand, the RobustGNG algorithm uses novel information-theoretic algorithmic techniques (based on

Minimum Description Length) to infer a state space without specifying a maximum number of iterations or handcrafted stopping criteria. This is useful for life-long incremental learning, which is a precondition for developmental robotics. On the other hand, the improved CrySSMEx is also an information-theoretic method which applies conditional entropy measures to derive a dynamic probabilistic model of states and transitions which can accurately but minimally describe the sensorimotor space. Furthermore, it is useful for deriving models at different (e.g. higher) levels of abstraction, which is useful for concept learning.

With respect to previous strongly related works [30, 29], where offline probabilistic models for density estimation of object poses are used, our work makes progress in the direction of inference of dynamic models based on probabilistic state machines, which is a precondition for planning. Moreover, as already mentioned, our work also focuses on incremental and active approaches for learning, which are preconditions for developmental robotics. Last but not least, the accuracy of these models can be relaxed to obtain coarse-grained models for categorization of object behaviours (concept learning methodology), which is also an additional layer upon the learning approaches we mentioned before.

The robot interacts with simple geometrical objects via a pushing action. After performing an action, a quantization function for density estimation is applied to a representation of the sensorimotor space in an incremental manner. In contrast to [30], which applied an offline method for density estimation of object poses, here we additionally apply incremental and active methods. These methods are described in Chapter 4. Moreover, the construction of qualitative models which can keep track of the temporal causality for long data sequences is an additional contribution of our work (see Chapter 5).

Basically, sequences of finger and object poses (rigid body transformations) are stored as training instances of the sensorimotor space. The functions are gradually refined, after new training sequences are gathered. The actions selected by the robot are possibly chosen by means of an active procedure according to an information-theoretic error measure. Then, the quantized space is split into two subspaces or regions after some iterations. This divide-and-conquer approach accelerates the convergence and induces more simple and tractable models for each region. We present a modification of the Growing Neural Gas algorithm [16, 49, 50, 53] for quantization which is robust for finding the right clusters in the presence of noise. In our algorithm, some modifications are intended to transform the algorithm towards a completely online procedure. In Fig. 4 the flow of information when quantizing sensorimotor sequences is presented. The diagram corresponds to both incremental and active cases, where the former corresponds to the random selection of actions.

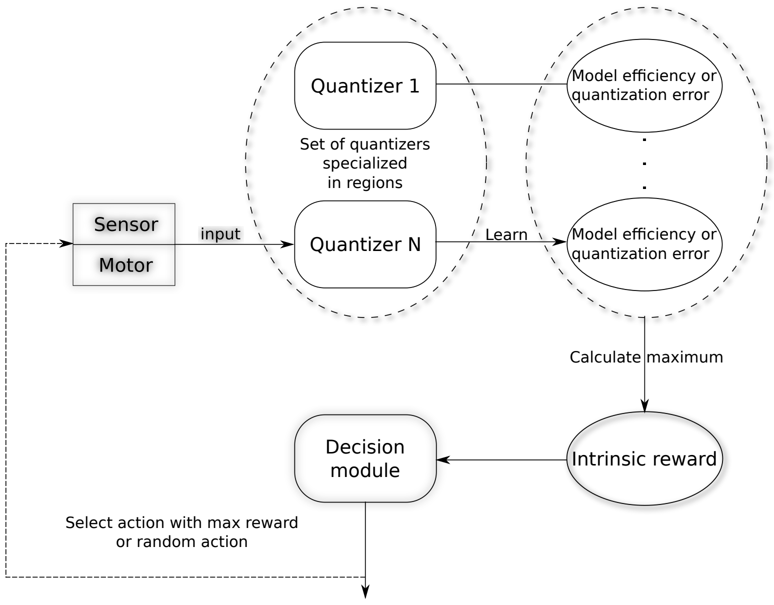

Quantizer 1
Set of quantizers specialized in regions
Quantizer N
Model efficiency or quantization error
Model efficiency or quantization error
Sensor
Motor
input
Learn
Calculate maximum
Intrinsic reward
Decision module
Select action with max reward or random action

Figure 4: Quantization schema. Quantization of input spaces via active sample selection.

RobustGNG is robust to noisy data. This is particularly important for our problem, specially in the context of real scenarios, where the information of object pose is derived from an object tracker, which produces a noisy ground truth. There is also uncertainty in the pose measurements from the robotic arm.

When the quantization process finishes, we employ an offline mechanism for constructing probabilistic models of action/object complexes that makes use of the quantization functions. They have the potential to be used for planning, since we obtain a qualitative representation of the sensorimotor space which encodes action/behavior instances. This method is based on the CrySSMEx algorithm [25, 53] for extracting substochastic sequential machines (SSMs) from dynamical systems. The robot predicts the object behavior in terms of trajectory information contained in sequences of object poses, given an arm pose and a certain action, represented as the target goal of the robot finger. The probabilistic models that we obtain are substochastic finite-state models. Moreover, their graph-based nature encodes the probabilistic transitions that lead to subsequent states, which is particularly useful in planning. In the experiments we present in this dissertation, the state space encompasses geometrical representations of object poses in 3-dimensional space.

The substochastic machines are represented as a special case of Mealy or Moore machines, which also contain probabilistic information. These machines allow us to encode a quantization of the state space, and to encode input functions and output functions, which are quantization functions. In this problem, input functions are a quantization mechanism applied to the representation of the sensorimotor space as mentioned above. They encode a combination of object/finger pose information. On the other hand, output functions may be used in different manners, depending on the discretization degree that we expect in the probabilistic finite-state machine. More precisely, if the objective were to accurately predict the trajectories of objects (in terms of object poses), given some action and some object pose, we could apply an output function that would act as a probabilistic regression, as long as we represent object poses at subsequent states in this output function.

In our work, we did not use this kind of output function, but rather a quantization mechanism on vectors of object transformations, to further reduce the space complexity of output quantizers. This output representation also takes advantage of the information-theoretic learning properties of CrySSMEx and allows for prediction of object trajectories. On the contrary, if we want to group state clusters on the basis of more coarse-grained sets (more abstract patterns), we might want to use other quantization functions. In our experiments, we obtained classifications for three different types of object behaviors by means of an alternative output quantization. We only evaluated these coarse-grained classifications in simulated scenarios to avoid additional complexity introduced by noisy data. A general learning schema is presented in Fig. 5, where different colors are used to identify space types.

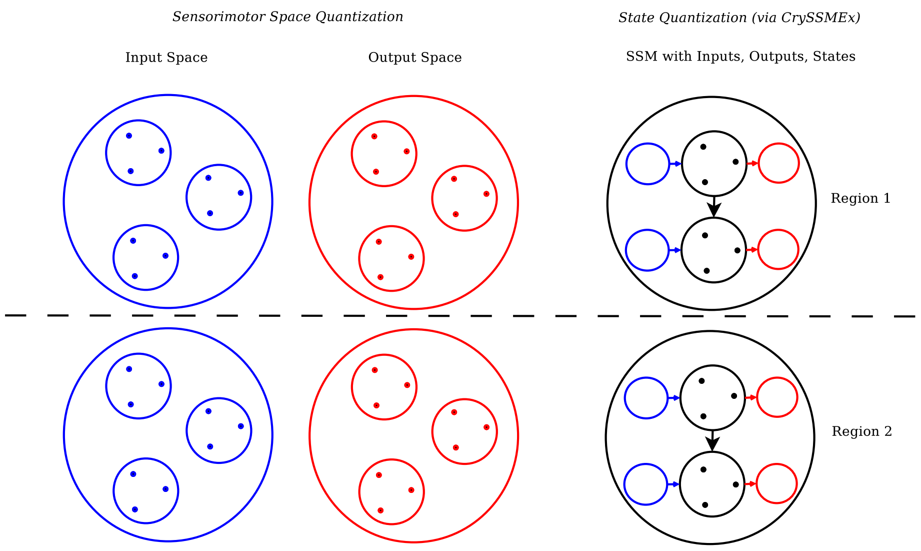

Sensorimotor Space Quantization
State Quantization (via CrySSMEx)
Input Space
Output Space
SSM with Inputs, Outputs, States
Region 1
Region 2

Figure 5: Learning schema. Quantization of input, output and state spaces and division of the spaces into regions.

Our learning approaches can also be discriminated in three models: offline, online, incremental and active learning. The information flow depending on which of these models is used is illustrated in Fig. 6.

Our learning scenarios involve a robotic arm, a polyflap (3-D object, cf. Chapter 3) in the simulated scenario and a box in the real scenario, as illustrated in Fig. 7. We used in our experiments only one object due to limited time and resource constraints such as hardware limitations but also to easily illustrate algorithmic issues associated to incremental and active learning, as we explain later in the experimental results. However, experiments with different objects and different shapes is an obvious extension of our research.

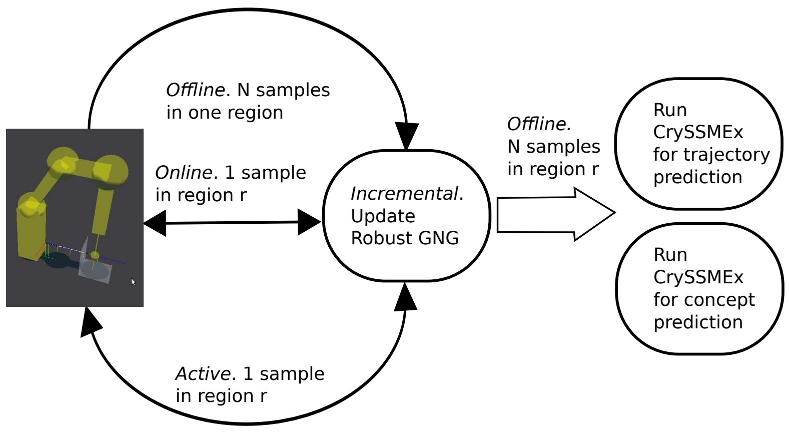

Offline. N samples in one region
Online. 1 sample in region r
Incremental. Update Robust GNG
Offline. N samples in region r
Run CrySSMEx for trajectory prediction
Run CrySSMEx for concept prediction
Active. 1 sample in region r

Figure 6: Information flow regarding different learning models.

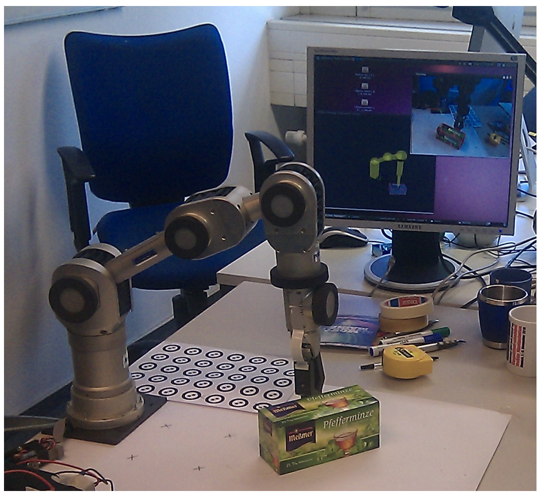


Figure 7: Learning scenario where a Katana robot pushes a tea box.

1.3 OVERVIEW

This dissertation is organized as follows. In Chapter 2, we review related work. In Chapter 3 we describe the features used in our learning experiments. In Chapter 4, we report on a study about the quantization mechanisms implemented and their application to synthetic data sets. Afterwards, we explain the process of quantizing sensorimotor spaces in a pushing scenario. In Chapter 5 we describe the process of inducing substochastic sequential machines. In Chapter 6 we analyze experimental results for prediction and classification after inducing SSMs. In Chapter 7 we close this dissertation with conclusions and discussion.

2 BACKGROUND

In this chapter we will discuss several topics which are relevant for the problem stated in the previous chapter. The solutions implemented in this dissertation are inspired or based on the subjects presented below. In the first section (2.1) we explain approaches for learning inspired by neuroscientific research, namely developmental robotics and active learning, essential for designing autonomous intelligent robots. Section 2.2 presents relevant literature about concept learning, i.e., methods to find out abstract patterns which are meaningful for humans from learning features. In Section 2.3 we describe meta-learning aspects to categorize and contextualize the different learning strategies used in this work in a broad sense. Next, in Section 2.4 we present theoretical aspects of quantization strategies, which is one of the key features of our learning algorithms for compressing information. Then, in Section 2.5 we describe approaches for sequence learning which are related to the extraction of probabilistic finite state models for dynamic interactive systems. Finally, in Section 2.6 we present preliminary results of our work, applying different but related strategies for solving the problems we are tackling.

2.1 DEVELOPMENTAL ROBOTICS AND ACTIVE LEARNING

As we mentioned in the previous chapter, this dissertation follows research directions led by several works in the Developmental Robotics field.

In [63], the author proposes five basic principles that describe the field of developmental robotics:

1. *Verification*: "An AI system can create and maintain knowledge only to the extent that it can verify that knowledge

itself" [63]. Thus, learning does not happen without verification.

2. *Embodiment*: since verification does not happen in the absence of actions, the robot must have a means of affecting the world, i.e., it must have a body.

3. *Subjectivity*: since a robot autonomously learns and verifies its knowledge, learning is a function of what the robot has experienced through its own sensors.

4. *Grounding*: successful verification, i.e. grounding, is achieved through the coupling of actions and their observable outcomes. It consists of *act-outcome* (or *behaviour-observation*) pairs.

5. *Incremental development*: exploration for learning is guided by an attention mechanism that can decide what parts of the environment are still interesting, i.e., when the levels of verifiability are still not perfect.

In the next paragraphs we briefly introduce some developmental robotics scenarios taken from different sources which follow these principles in general terms. In our experiments, the verification and grounding principles are considered during testing (estimation, prediction or classification ability) of the learning algorithms, as we will see in Chapters 4 and 5. Incremental development is achieved by applying incremental and active density estimation methods of sensorimotor spaces, and the construction of models for prediction of behavior/observation tuples.

In [68], a robotic agent with social skills but limited manipulation skills uses socially guided exploration for solving simple tasks by applying the following strategies (see Fig. 2):

- Coupled interaction with a human teacher, where the teacher is able to infer the current learning state of the robot through robot demonstrations of the current task and nonverbal social cues like emotions.

- Communication or feedback from the teacher to attain the goal of the current task.

The robot is equipped with a self-motivated exploration system and a limited set of actions. Self-motivated exploration can have different motivation sources. The following drives for a robot were identified in [68]:

- Activity drive, which reflects the current level of activity.
- Novelty drive, measuring how novel recent events have been.
- Mastery drive, measuring the level of confidence of the current system state.

The selection of an action is not only driven by one of the aforementioned motivation sources but also from task-relevant actions. In that way, the robot and the tutor are expected to work as a team to achieve a common goal. The mechanism for learning how to solve a task is based on reinforcement learning strategies, in particular *task option policies* [68]. The name was chosen by the authors to reflect the similarities with the well-known *options* framework from the reinforcement learning literature [67].

In contrast, in [47] a more autonomous agent performs actions on the basis of an intrinsic motivation system inspired by psychological and neuroscientific discoveries, as seen at the beginning of this section. The authors demonstrate experimentally that the robot enters into different stages of development during exploration steps where actions are selected according to a measure of learning progress. In that way, the robot carries out novel actions and seems to be curious, because it mostly selects actions which maximize the probability of increasing the learning progress calculated from previous and current error estimates in predicting the action. In Fig. 8 a setup of this playground experiment is shown. This approach is inspired by reinforcement learning techniques where a reward for performing an action exists [73, 66]. In this case, however, the reward is intrinsic (self-motivated). This algorithm is called Intelligent Adaptive Curiosity (IAC). Previous works had implemented agents with artificial curiosity and novelty drives, based primarily on reinforcement learning approaches [57, 69, 28, 13].

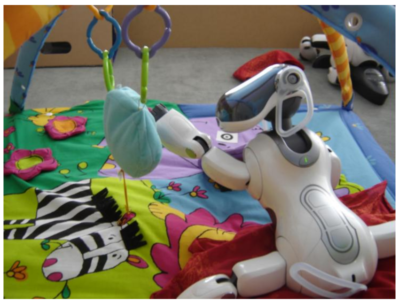


Figure 8: The playground experiment setup [47].

The work presented in this dissertation is also influenced by developmental strategies for evolving the prediction machinery of the robot based on IAC. In particular, [47] presented an experimentally rigorous analysis about the effects of introducing an intrinsic reward for selecting actions. They did not implement more complex learning methods as in [67] in order to reduce the complexity of the problem and focus only on analyzing the implications of such a reward in a developmental learning phase. Moreover, given the high dimensionality and complexity of real robotic environments, implementing reinforcement techniques remains a big challenge.

A nearest-neighbor algorithm was implemented for prediction. A sensorimotor context [47] is defined as a vector concatenation of the sensory $\mathbf{S}(t)$ and motor $\mathbf{M}(t)$ inputs at time t ($\mathbf{SM}(t)$). Sensorimotor regions are defined from the sensorimotor context and the predicted next step sensory input. Thus, an exclusive set of instances $\langle \mathbf{SM}(t), \mathbf{S}(t+1) \rangle$ are stored for a certain region. In each region, a corresponding learning expert is specialized. At the beginning of the developmental learning phase, there is only one region and it is recursively divided primarily on the basis of a variance criterion among the corresponding instances. During the learning process, the robot occasionally focuses on interesting and predictable events, such as biting or pushing objects. These events are correlated to the increase in learning progress at a certain learning step. In [47] the authors state that IAC could be applied for slowly learning machines like backpropagation neural networks. Our preliminary experiments [54] implemented an adapted version of IAC where a recurrent neural network is employed for prediction (see Section 2.6).

As pointed out in [47], the strategies for selecting actions efficiently are closely related to the field of active learning [12] in machine learning terminology, where the objective is to maximize the information gain when selecting the next example and thus reduce the amount of instances needed for concept learning.

Developmental phases can also be distinguished from the work in [42]. Their approach for robot learning takes place in a simulated scenario with individual objects where the states and actions are not defined explicitly (see Fig. 3). A system which can find out qualitative states and actions is designed instead. Firstly, the agent selects random motor values, i.e, the agent "motor babbles" for many iterations, after which it learns to predict and control the variables of the environment, defined as motor, magnitude and direction of change variables. *Landmarks* of variables which define discrete states help find discrete events. Candidate landmarks are for instance discovered by an information-theoretic criterion which assesses the information gain among events. The relationships among events are used to learn rules (models of causal relationships) encoding actions and states by using dynamic Bayesian networks (DBNs) [42]. Before learning and during qualitative state inference, the agent usually has to decide which actions are relevant, in which case the active learning method [47] described above is employed. The models learned by the DBNs are then transformed into Markov decision processes (MDPs) to be used for planning by using the options framework, so that robots learn a given task [42].

A developmental approach was also followed in [32] for acquiring object and grasping knowledge. Different types of objects are considered, as seen in Fig. 9. In that work, the robot performs exploratory actions and refines its models gradually, during a developmental learning process with different stages. In the first stage, the robot learns visual models, and in the second it learns object-specific grasp affordances [32]. A third stage involves the use of the acquired competences to execute a plan for grasping. More concretely, a set of actions aimed for grasping are triggered by specific 3-D feature sets in the first stage, storing in an episodic memory the outcome (failure or success in grasping) and associated relationships among visual features and proprioceptive information (grasping affordance). At the end of this stage, this set of features represent the objectness or object shape. Then, in the second stage, a pose estimation algorithm, combined with grasping affordances obtained beforehand, is useful for obtaining a grasping memory associated to an object. Finally, planning with grounded objects and grasps is carried out making use of the OAC formalism [48].

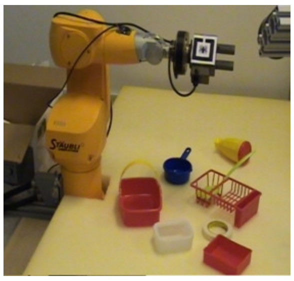


Figure 9: Experimental setup for grasping through exploration [32].

2.2 CONCEPT LEARNING

Previous work on affordances learning [55] has presented a survey on formalizing the concept of affordance and proposed an extended definition:

Definition 1 *An affordance is an acquired relation between an acquired* effect *and a tuple* (entity, behavior) *such that when the* agent *applies the* behavior *on the* entity, *the* effect *is generated. An affordance is then a relation* (effect, (entity, behavior)).

In [72], the authors implement the above formalism. The robot interacts with objects and memorizes the triples obtained to be further used in a planning framework. The work in [3] shows the relationship among object concepts and verb concepts with affordance relations that a robot acquires through interaction with objects. They also use the abovementioned formalization and perform two alternative methods for categorization. First, they found clusters in a space $\langle e_1, \ldots, e_i \ldots, e_n \rangle$, where e_i is the predicted effect of the i_{th} behavior on the object. Secondly, a clustering algorithm was applied to the effects of the behaviors, by applying a feature selection method. For instance, a system learns the behavior "push left" and discovers that the entities that have led to the effect category "pushed left" and "rolled left" are distinguishable according to some feature element f_1 [3].

In [51, 52] a similar approach for recognizing affordances was derived. The experimental setup consists of a tabletop scenario with a robotic arm which interacts with pushing actions on flat-surfaced and curve-surfaced objects (see Fig. 10). Visual object features like shape features or global motion features are obtained in order to train an unsupervised clustering algorithm in the space of features. Then, either unsupervised or supervised vector quantization algorithms such as Learning Vector Quantization (LVQ) are used for the classification of affordances such as rolling and non-rolling.

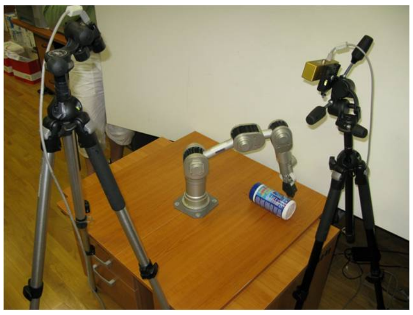


Figure 10: Affordance learning scenario [51, 52]

It is clear that concept learning or affordance learning has a tight connection with the pattern recognition framework, whether done in a supervised or unsupervised manner. In our work, we explore in Chapter 5 a pattern recognition process involving the dynamics of object behaviors, which was not considered in the work described above. We argue that our approach allows the construction of models of causal relationships which are useful for analyzing and understanding complex dynamical systems, without the need for predefining specific visual feature sets. Moreover, as we state in Chapter 1, these models can then be useful for planning (control) by querying them to verify the acquired sensorimotor concepts in an efficient way, since the rules governing the dynamical systems are verifiable in the extracted models.

The *object-action complexes* (OAC) framework [36] provides a similar theory which makes use of the concept of affordance which enables planning and execution of actions. In this formalization, the OAC theory defines affordances more generally as state-transition functions suited to prediction. An agent performing an action to achieve some effect will have to know about the attributes necessary to execute the action. More formally,

Definition 2 *An OAC is a triplet* $(id, \mathsf{T}, \mathsf{M})$*, where id is a unique identifier,* $\mathsf{T} : \mathsf{S} \rightarrow \mathsf{S}$ *is a prediction function where* S *is a global attribute space and* M *a statistical measure representing the success in prediction of the OAC over the past.*

This framework has been used for the acquisition of grasp and push affordances. In the case of grasping, a state is defined as a stable grasp, a collision, a non-successful grasp or a non-stable grasp. For pushing, initial and final object poses after a pushing action define the states. In comparison with our work on pushing actions (Chapters 4 and 5), we also consider information about behaviour dynamics, in contrast to the framework presented above, where the state space is discretized by the prediction function defining the system states.

2.3 METALEARNING

2.3.1 *Learning methods*

In robot learning it is essential to come up with learning models which can be immediately (incrementally) updated after new information arrives. This precondition imposes requirements such as proper generalizations (robustness) and processing efficiency.

Here we follow the distinction made by [18] around different learning methods, extending it with the concepts of active and passive learning [69] (see Table 1).

Learning Method	Description
Batch	After initializing the model, the training data are completely processed before the model is updated. This is repeated until a certain criterion is met. An iteration is also called an epoch.
Incremental	After initializing the model, in each iteration only one training example is processed and then the model is updated.
Offline	All the data are given in advance. The stored samples can be accessed repeatedly. Therefore, batch learning is always offline.
Online	Each training sample is discarded after it has been processed and the model is updated. Online learning is always incremental but incremental learning can be done online or offline.
Passive	A stream of training data is generated by the environment according to some unknown probability distribution. All the above methods can be applied for passive learning.
Active	In contrast to passive learning, where the learning method is a pure observer, in active learning the learner has the ability to interact with the environment, in order to generate new samples. More specifically, the learning method can execute actions which generate new training data. Thus, here it is important to come up with measures of efficiency for the selection of actions. Since the learners generate the samples, this is another case of incremental learning (which could be performed online or offline).

Table 1: Summary of learning methods.

In our work, apart from offline approaches to learning, we also focus on incremental and active learning methods.

2.3.2 *Lifelong learning*

During their entire lifespan, human beings are able to learn and face different learning problems. They have to acquire different learning skills to achieve goals. Learning patterns, communication, logical reasoning and language, among others, are different aspects of a lifelong learning process [69]. There are opportunities to transfer knowledge between learning tasks. For instance, humans can extract relevant features to generalize a complex target concept well even if the number of potentially relevant features is huge. This ability relies on previously learned knowledge, acquired earlier in the lifetime of an individual [69]. Thus, concept learning, knowledge transfer and incremental learning are also elements of this lifelong learning context. This formalization of learning has similarities to the developmental learning approach in the sense that different learning tasks and transfer of knowledge between tasks arise during and after developmental phases.

2.4 QUANTIZATION

2.4.1 *Terminology and Definitions*

Robots need to cope with large amounts of information which must be compressed in an efficient way. The process of inferring the underlying structure of a data set can be approached in different ways. Different terminology and methods have been also applied in order to obtain qualitative representations of data sets.

One particular problem is the grouping of similar data items from a set of input vectors $\mathbf{x}$ in clusters, called *clustering*. This problem is strongly related to *density estimation*, which aims to determine the distribution of data within the input space [6, 21]. In these tasks, there is no information available about the target values or classes associated to some cluster.

Clustering is a term originating in the pattern recognition literature. It is also called unsupervised classification. The problem in clustering is to group a given collection of unlabeled patterns into meaningful clusters, in contrast to supervised classification, where the given (labeled) patterns are used to learn the descriptions of classes which in turn are used to label a new pattern. In clustering, labels are associated with clusters but these category labels are data driven (obtained solely from the data) [26].

A clustering task usually comprises four main steps: a pattern representation step which might include feature extraction or selection, the definition of a pattern proximity measure appropriate to the data domain, clustering and data abstraction if needed [26]. Fuzzy clustering techniques assign to an input pattern a fractional membership degree in each cluster, in contrast to hard clustering where a specific class label is assigned to each pattern, identifying a class [26]. To solve the clustering problem, different machine learning and pattern recognition algorithms have been employed, including fuzzy clustering, neural networks, kernel density estimation, nearest-neighbors, principal component analysis and self-organizing maps, among others.

On the other hand, the term *quantization* originated in the signal processing literature as a process of discretizing signals. In [19] it is regarded as a tool for clustering or cluster analysis for quantization of empirical probability measures. As a mathematical topic it can be defined as follows [19]:

Definition 3 *"Quantization for probability distributions concerns the best approximation of a* d*-dimensional probability distribution* P *by a discrete probability with a given number* n *of supporting points or in other words, the best approximation of a* d*-dimensional random vector* X *with distribution* P *by a random vector* Y *with at most* n *values in its image."*

Finally, the term density estimation is a statistical concept. The problem refers to the modeling of a probability distribution $p(\mathbf{x})$ of a random variable $\mathbf{x}$, given a finite set $\mathbf{x}1, \dots, \mathbf{x}_N$ of observations [6]. A common probability distribution used for density estimation is the Gaussian distribution. Introducing discrete latent variables enables the modeling of more complex multimodal distributions in the form of mixtures of Gaussians. A discrete latent variable may have discrete binary values representing the relationship between a data point $\mathbf{x}_n$ and a latent variable $\mathbf{z}_k$ associated to the component k of the Gaussian mixture. A well-known method for finding maximum likelihood solutions for models with latent variables is the expectation-maximization (EM) algorithm [6]. Whereas in hard clustering algorithms each data point is associated uniquely with one cluster, the EM algorithm makes a soft assignment based on the posterior distribution of latent variables, given the observed variables and distribution parameters [6].

2.4.2 *Model Selection*

The problems above mentioned concern the ability of a learning machine to find a good explanation for the distribution of a data set. In this work, we use information-theoretic techniques for finding proper models or hypotheses. A good model is one that avoids *overfitting*, offering a good generalization for, e.g., prediction, pattern classification and parameter estimation. Finding this model is known as the model selection problem.

Overfitting occurs when a model consists of many parameters which fit the known data set well, but it does not offer good prediction performance. This absence of good generalization ability (robustness) is an important issue in inductive and statistical inference [22]. In Bayesian inference [6], the overfitting associated with maximum likelihood, which is a common frequentist estimator, can be avoided by using a Bayesian approach involving probabilities to represent uncertainty in the choice of a model. Thereafter, models are compared directly on the training data. Another common approach is the use of cross-validation, where some proportion of the data set is used for training while using all data to assess performance. Other information-theoretic approaches like the Bayesian information criterion have been applied to correct for the bias of maximum likelihood [6].

In order to compare different explanations (quantizers) for our data sets, we employ the information-theoretic mechanism called the Minimum description length (MDL) principle. This inductive inference method provides a generic solution to the model selection problem, and, more generally, to the overfitting problem. MDL is based on the following insight: any regularity in the data can be used to compress the data, i.e., to describe it using fewer symbols than the number of symbols needed to describe the data literally [22]. This idea leads to a formalization of a theory of inductive inference with the following properties, among others [22]:

- *Occam's razor.* MDL chooses a model that trades off goodness-of-fit on the observed data with "complexity" or "richness" of the model.
- *No overfitting.* MDL methods automatically and inherently protect against overfitting and can be used to estimate both the parameters and structure of a model.
- *Predictive interpretation.* Data compression is formally equivalent to a form of probabilistic prediction. Thus, MDL methods search for a model with good predictive performance on unseen data.

MDL has been used for model selection, prediction, parameter selection, parametric and non-parametric density estimation, and in general in every type of inductive inference task such as denoising, similarity analysis and clustering, outlier detection and transduction [22].

For robust vector quantization, the MDL principle was used in [4]. The problem of vector quantization is defined there as follows [4]:

Definition 4 *Given a finite data set* $S = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^d$ *with* $d \in \mathbb{N}$, *where* $\mathbf{x}_i$ *are independently and identically distributed (iid) according to some probability distribution* $p(\mathbf{x})$, *find a set of reference vectors* $A = \{\mathbf{c}_1, \dots, \mathbf{c}_m\}$, $\mathbf{c}_i \in \mathbb{R}^d$ *such that a given distortion measure (i.e. expected quantization error)* $E(p(\mathbf{x}), A)$ *is minimized.*

Since the only information available is the data set S, the objective is to minimize the expected quantization error

$$E(S, A) = \sum_{i=1}^{m} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mathbf{c}_i\|^2, \tag{1}$$

where $S_i = \{\mathbf{x} \in \mathbb{R}^d | i = \arg\min_{j \in \{1,\dots,m\}} \|\mathbf{x} - \mathbf{c}_i\|\}$ is the Voronoi region of a vector $\mathbf{c}_i$ (cf. Fig. 11). This minimization is achieved by adequately positioning the vectors $\mathbf{c}_i$ [4].

The authors approach the problem as a minimization of the description of the training data S. Usually, a description is given in terms of a coding system [22]. In the formalization

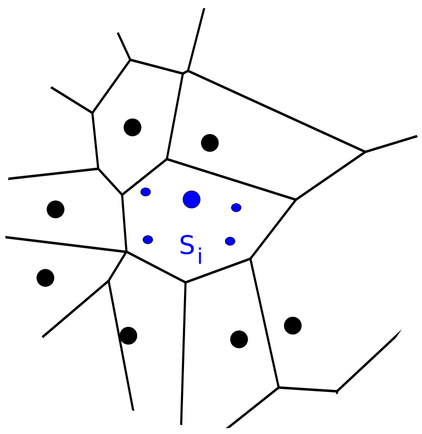


Figure 11: Voronoi region S_i.

given by [4], the goal is to find a set of reference vectors[1] A to encode S optimally. Some data vectors O are considered outliers and another set as inliers I, thus $I = S - O$. Using A, the length of encoding S is then given by [4]:

- The length of encoding the reference vectors A, $L(A)$.
- The length of encoding I using A, which is subdivided into two costs:
  - The length of encoding the index of A to which the vectors in I have been assigned, $L(I(A))$.
  - The length of encoding the residual errors, $L(\epsilon(I_A))$
- The length of encoding the outliers, $L(O)$.

Thus, the cost of encoding S using A is given by [4]

$$L(S(A)) = L(A) + L(I(A)) + L(\epsilon(I_A)) + L(O). \tag{2}$$

The goal is then to minimize $L(S(A))$, i.e., determine O, m and $\mathbf{c}_i, 1 \leqslant i \leqslant m$ such that $L(S(A))$ is minimal. Assuming that these quantities are specified with a finite precision η and that reference vectors are represented by K bits, Eq. 2 can be reformulated as Eq. 3 [4]:

$$L(S(A)) = mK + L(I(A)) + \sum_{i=1}^{m} \sum_{x \in S_i} L(\mathbf{x} - \mathbf{c}_i) + |O|K. \tag{3}$$

1 which can also be called model vectors, prototype vectors or centroids

In general, any clustering algorithm may be used to adapt the reference vectors. In this method [4], a vector quantization network is initialized with a high amount of reference vectors, after which unnecessary reference vectors are removed and outliers are detected, improving the network quality in terms of MDL. An advantage of this method with respect to well-known clustering techniques such as K-means and Gaussian mixture models [6] is that the number of reference vectors m is not given a priori. Indeed, MDL is used here to select among different models of a data distribution containing different components.

2.4.3 *Growing Neural Gas for Quantization*

The algorithm Growing Neural Gas (GNG) [16] was conceived for unsupervised learning, where there is no available information about output classes. GNG is potentially useful for quantization, yet still needs a method for finding the optimal reference vectors. To solve this, combining GNG with MDL has been proposed, as we explain below. The algorithm initially targeted a problem called *topology learning*: given some probability distribution, the objective is to find a topological structure which closely reflects the topology of the data distribution, represented by a set of centers (reference vectors). For this purpose, a competitive Hebbian learning (CHL) method is combined with the vector quantization method Neural Gas [38]. The main advantages of the Neural Gas model according to [40] are:

1. Faster convergence to low distortion errors.
2. Lower distortion error than that resulting from K-means clustering, maximum entropy clustering and Kohonen's self-organizing feature map.
3. Obeying a stochastic gradient descent on an explicit energy surface.

Assuming there are a number of centers in $\mathbb{R}^n$, CHL successively adds topological connections among them. Given a data sequence $\mathbf{x}$ drawn from some data distribution $p(\mathbf{x})$, the principle of CHL is [38]: "for each input $\mathbf{x}$ connect the two closest centers (measured by Euclidean distance) with an edge." The resulting graph is a subgraph of the Delaunay triangulation which has been shown to optimally preserve topology in a general sense [38].

To make use of all the centers, they have to be placed in regions where $p(\mathbf{x}) > 0$. This can be achieved by any vector quantization algorithm. In [16], the Neural Gas method was used for that purpose, with the principle: "for each input $\mathbf{x}$ adapt the k nearest centers whereby k is decreasing from a large initial to a small final value".

Since the motion of centers might make edges which have been generated earlier invalid, an edge aging scheme is also considered for the purpose of removing obsolete edges. Combining CHL and Neural Gas is a proper method for preserving topology as proven in [39], but the number of centers has to be decided a priori, which is a common problem in vector quantization algorithms. The GNG algorithm proposes an incremental solution to this problem, but a maximum a priori number of nodes or some performance measure is still needed as a stopping criterion.

A GNG network consists of [16]:

- A set A of units (nodes). Each unit $c \in A$ has an associated reference vector $\mathbf{w}_c \in \mathbb{R}^n$. The reference vectors can be regarded as positions in the input space of the corresponding units.
- A set N of connections (edges) among pairs of units. These are not weighted and their sole purpose is to define topological structure.

The main idea of the method [16] is to successively add new units to an initially small network by evaluating local statistical measures gathered during previous adaptation steps. In summary, the GNG algorithm executes the steps illustrated in Algorithm 21 [16].

The adaptation steps towards the input samples (lines 8,9) lead to a general movement of all units towards areas of the

Algorithm 1 : GNG

```
1  begin
2      Start with two units at random positions;
3      while A stopping criterion (e.g. net size or some performance measure) is not yet fulfilled
       do
4          Generate an input sample $\mathbf{x}$ from the data distribution $p(\mathbf{x})$;
5          Find the nearest unit $s_1$ and the second nearest unit $s_2$;
6          Increment the age of all edges emanating from $s_1$;
7          Add the squared distance between the input sample and the nearest unit in
           the input space to a local error counter variable: $\text{error}_{s_1} = \|\mathbf{w}_{s_1} - \mathbf{x}\|^2$;
8          Move $s_1$ and its topological neighbors towards $\mathbf{x}$ by fractions $\epsilon_b$ and $\epsilon_n$,
           respectively, of the total distance: $\Delta\mathbf{w}_{s_1} = \epsilon_b(\mathbf{x} - \mathbf{w}_{s_1})$;
9          $\Delta\mathbf{w}_n = \epsilon_n(\mathbf{x} - \mathbf{w}_n)$ for all direct neighbors $n$ of $s_1$;
10         if $s_1$ and $s_2$ are connected by an edge then  Set the age of the edge to zero;
11         ;
12         else if such an edge does not exist then
13             create it;
14         Remove edges with an edge larger than $a_{max}$. If this results in points having
           no emanating edges, remove them as well;
15         if the number of input samples generated so far is an integer multiple of a parameter $\lambda$
           then
16             Insert a new unit as follows;
17             Determine the unit $q$ with maximum accumulated error;
18             Insert a unit $r$ halfway between $q$ and its neighbor $f$ with the largest
               error variable $\mathbf{w}_r = \frac{1}{2}(\mathbf{w}_q + \mathbf{w}_f)$;
19             Insert edges connecting the new unit $r$ with units $q$ and $f$, and remove
               the original edge between $q$ and $f$;
20             Decrease the error variables of $q$ and $f$ by multiplying them with a
               constant $\alpha$. Initialize the error variable of $r$ with the new value of the
               error variable of $q$;
21             Decrease all error variables by multiplying them with a constant $d$;
```

input space where $p(\mathbf{x}) > 0$. The insertion of edges (line 10-12) between the nearest and the second-nearest unit generates a connection of the "induced" Delaunay triangulation, whereas in line 13 edges are removed which are no longer part of it. This is achieved by local edge aging (line 6) and age resetting (line 10) [16]. The accumulation of squared distances (line 7) helps to identify areas with high accumulated errors, which are reduced by adding units in these regions.

The GNG algorithm can potentially be used for quantization, provided some adjustments are applied. GNG was improved in subsequent work [49, 50] to deal with robust quantization of noisy data sets. In [49], an outlier resistant strategy, an adaptive learning rates and cluster repulsion scheme, and a criterion for determining the optimal number of clusters were proposed. The updating rule in GNG (lines 8,9 in Algorithm 21) is inherently fragile in noisy environments and sensitive to the order of input vectors [49].

The outlier-resistant strategy is based upon adding a parameter used to limit the updating strength caused by some outlier. In Section 4.1.1.1, specifically in Eq. 12, we can see a general form of the learning rule including an outlier-resistant parameter σ_k for some node k.

In [49] adaptive learning rates ϵ_b and ϵ_n were introduced. This strategy solves issues introduced by static learning rates in GNG, which prevent the refinement of the positions of current prototypes towards actual cluster centers. The adaptive learning rates take different values for nodes inserted in different order, and meanwhile make them decrease monotonically with the increment of nodes. These values depend on the predefinition of a maximum number of nodes. In our work, we apply a different strategy which depends less on a priori parameters and is explained in Section 4.1.1.1.

In [49] a repulsion scheme was also proposed, by adding repulsive forces to the updating rule such that two nodes do not try to approach one cluster simultaneously. In our work, we avoid the use of repulsive forces by adding a more efficient node insertion mechanism (Section 4.1.1.2).

Finally, in [49], the MDL principle is proposed for determining the optimal number of clusters. The authors instantiated the calculation of MDL as in [4] (explained above in Section 2.4.2). However, they still use a maximum predefined number of nodes or some predefined performance measure as a stopping criterion. The minimal MDL value found during learning helps to identify the network with the optimal cluster number. In contrast, we use MDL (Sections 4.1.1.3 and 4.1.1.5) to calculate a network stability measure to be used as a stopping criterion.

The work presented in [50] is a follow-up to the robust quantization mechanism already explained [49]. Here the authors employ the same methods to make GNG robust. However, they use MDL and the local error quantities to relocate nodes in more proper locations of the input space. This method requires many calculations. Therefore, we came up with more efficient strategies (Chapter 4) for dealing with computational complexity issues.

2.5 SEQUENCE LEARNING

In this dissertation we deal with learning problems where sequences of features are involved. Sequential data can arise in contexts like time series, natural language sentences, genetic sequences, financial data [6] or other types of dynamical systems. The sequence learning problem states that given past observations, we want to predict the next value in a time series. Since considering a general dependence of future observations on all previous observations would be intractable, a *Markov* assumption is often made, in which we assume that future predictions are independent of all but recent observations [6]. Although these Markov models are tractable, they are also limited. Therefore, state space models like hidden Markov models (with discrete latent variables) and linear dynamical systems (latent variables are Gaussian) have been introduced [6].

The problem of learning and predicting data sequences from a dynamical system is closely related to time-series prediction. An extensive literature review can be found in [11]. Different methods have been applied to solve these problems, including neural networks and recurrent neural networks [33]. As [45] mentions, most of the existing research on time series concerns supervised forecasting problems. However, in recent years exploration of unsupervised algorithms for analysis of time-series has taken place [45, 5, 1]. An example of these methods is the generative topographic mapping (GTM) through time (a linear dynamical system), which embeds a mixture model in a hidden Markov model (HMM). In [45, 5] the hidden states of a HMM correspond to latent variables in GTM. A different approach is taken in [1], where time series are predicted by using a recursive growing self-organizing network based on the Growing Neural Gas (GNG) algorithm. A recent approach is dynamic conditional random fields (DCRFs) [65] which are undirected graphical models which repeat their structure and parameters over a sequence of state vectors to represent hidden states and complex interactions among labels in a similar way as in dynamic Bayesian networks (DBNs) [43].

It is also worth mentioning that HMMs are equivalent to *probabilistic non-deterministic automata* [15]. In [25], the concept of substochastic sequential machines was introduced. Substochastic sequential machines resemble *stochastic sequential machines* or *probabilistic automata* but there is a possibility of model "incompleteness" due to a finite observed data set [25].

In [25] we find a taxonomy of methods for extracting finite-state representations from recurrent neural networks. For instance, in [71], stochastic machines are obtained by partitioning the RNN state space and calculating probability estimates to obtain probabilistic transitions. In [25], is is argued that CrySSMEx distinguishes from earlier rule extraction approaches in four principles:

- quantization of state,
- observation of the underlying system,
- rule construction,
- rule minimization.

CrySSMEx has been tested for inferring machines which model formal grammars learned by RNNs (mainly regular grammars but also context-free grammars). It has also been used in the analysis of a chaotic function. In general terms, CrySSMEx can either perform a quantization of the input and output spaces, or the quantization can explicitly be given in form of symbolic representations. Then, a mechanism based on the minimization of conditional entropy between pairs of inputs and states, and outputs or next states, is performed to construct the probabilistic automaton, gradually splitting the state space.

However, the quantization mechanism for input and output spaces in CrySSMEx is quite simple, being based on a regular partitioning of the spaces. This may lead to more complex machines, inconsistent ones, or simply longer learning times in the presence of noise. Therefore, as we already pointed out in Section 2.4.3, in this dissertation we present a new method for quantizing spaces in an unsupervised manner by means of an algorithm based on GNG (see Chapter 4).

The original CrySSMEx algorithm also has some drawbacks that prevent the extraction of optimal machines for dynamical systems with certain symmetrical features and long dependencies among input, state and output tuples across time. We carried out experiments with synthetic stochastic systems with such characteristics (see Section 6.1) and we improved CrySSMEx correspondingly to handle these issues. We describe these methods in Chapter 5 and in Section 6.2 we evaluate them in our robotic learning scenarios.

2.6 PRELIMINARY EXPERIMENTS

We carried out preliminary experiments for prediction of an object pose, given previous effector (finger) and object poses, as already stated in Section 1.1. We employed a recurrent neural network (RNN) to solve the prediction problem as a regression task. Specifically, we use the Long Short-Term Memory (LSTM) [20] model of an RNN. The learning scenario with simulated objects is shown in Fig. 15 (Chapter 3). The features corresponding to the arm are a starting 6-D pose vector for the end-effector $\mathbf{e}_0$, and a real value denoting a direction angle Θ ranging from 60 to 120 degrees, parallel to the ground plane in the direction to the center of the standing polyflap side. Together, these features form the motor command feature vector denoted as $\mathbf{m}$. The values are all normalized to obtain vectors with mean 0 and standard deviation 1.0. A 6-D pose vector corresponding to the polyflap pose is denoted as $\mathbf{p}_t$ at time t. The pose $\mathbf{p}_0$ is fixed for all experiments.

Then, the concatenation $\mathbf{f}_0 = [\mathbf{m}\ \mathbf{e}_0\ \mathbf{p}_0]$ represents the feature vector to be fed initially to the neural network. The subsequent feature vectors fed to the machine have the form $\mathbf{f}_t = [\mathbf{o}\ \mathbf{e}_t\ \mathbf{p}_t]$, where the size of $\mathbf{o}$ is the size of $\mathbf{m}$. This representation allows the learning machine to attain a better convergence.

During the execution of the arm path, we obtain a series of poses $\langle \mathbf{p}_t, \mathbf{e}_t \rangle$ to construct a feature vector $\mathbf{f}_t$. We then extract n polyflap- and effector poses and finally we build a sequence set

$S = \{\mathbf{f}_{t=1}^{n}\}$. Thus, a particular sequence set (an instance) is used in each iteration of the experiment, to be fed to the LSTM in $n+1$ steps. For the time step t, a training tuple $\langle \mathbf{f}_t, \mathbf{t}_t \rangle$ is used for the neural network learning procedure, where the feature vector $\mathbf{f}_t$ represents the input vector and $\mathbf{t}_t = \mathbf{p}_{t+1}$ the target (predicted) vector encoding the predicted polyflap pose.

This representation then encodes the rigid body transformations of polyflap and effector through these n steps and also encodes the given robot control command that performs the pushing movement. In order to discretize and reduce the dimensionality of the task, we only used a discrete number of different starting positions for the arm to start the pushing movement.

As mentioned above, a dataset $\mathcal{D}$ containing a certain quantity of sequences S_i is obtained and we perform offline experiments with these data.

An LSTM machine is usually composed of an input layer, a hidden layer and an output layer. In general, recurrent neural networks can have recurrent connections for all their neurons. In this work we only use recurrent connections for the hidden layers. We also performed preliminary experiments with networks having no recurrent connections and these exhibited decreased performance, since sequences usually involve time dependencies which are represented in an LSTM by recurrent connections. The LSTM [20] architecture was developed in order to solve some learning issues in recurrent neural networks related to long-term dependency learning. These problems sum up to the problem that errors propagated back in time tend to either vanish or blow up. This is known as the problem of vanishing gradients.

LSTM's solution to this problem is to enforce *constant* error flow in a number of specialized units, called constant error carrousels (CECs), corresponding to those CECs having linear activation functions not decaying over time. CECs avoid transmitting useless information from the time-series by adding other input gates that regulate the access to the units. Thus, they learn to open and close access to the CECs at appropriate moments. Likewise, the access from the CECs to output units is controlled by multiplicative output gates and they learn in a similar way how to open or close access to the output side. Additionally, forget gates [20] learn to reset the activation of the CECs when the information stored in them is no longer useful, i.e., when previous inputs need to be forgotten. The combination of a CEC with its associated input, output and forget gate is called a memory cell, as depicted in Fig. 12. Other additions are peephole weights, which improve the LSTM's ability to learn tasks that require precise timing and counting of internal states, and bidirectional connections [20].

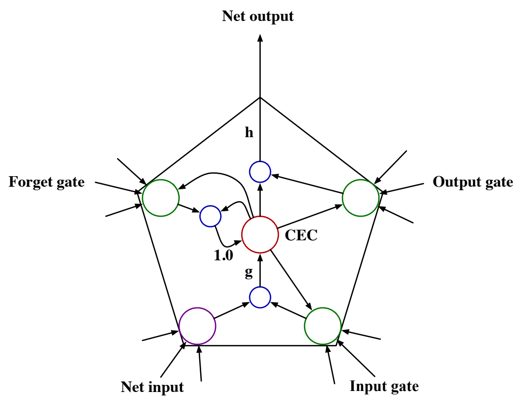


Figure 12: LSTM memory block with one cell. The internal state of the cell is maintained with a recurrent connection of fixed weight 1.0. The three gates collect activations from inside and outside the block, and control the cell via multiplicative units (small circles). The input and output gates scale the input and output of the cell while the forget gate scales the internal state. The cell input and output activation functions (g and h) are applied at the indicated places [20].

In this work, we used 10 memory blocks in the hidden layer, which was found to be a good compromise between computational complexity and convergence.

When some input vector is fed to the network, the forward pass is calculated as follows. Let us denote an output neuron (unit) activation y^o, an input gate activation y^{in}, an output gate activation y^{out} and a forget gate activation y^f. Then, for the time step t, each of them are calculated in the following standard way:

$$y^i(t) = f_i(\sum_j w_{ij} y^j(t-1)), \tag{4}$$

where w_{ij} is the weight of the connection from unit j to unit i, and f the activation function. We only consider one CEC activation (one cell) for each memory block. The CEC activation s_c for the memory cell c is computed as follows:

$$s_c(t) = y^{f_c}(t)s_c(t-1) + y^{in_c}(t)g(\sum_j w_{cj}y^j(t-1)), \qquad (5)$$

where g is the cell input activation function. The memory cell output is then calculated by

$$y^{s_c}(t) = y^{out_c}(t)h(s_c(t)), \qquad (6)$$

where h is the cell output activation function. The backward pass is a steepest (gradient) descent method which updates the weights of the different types of units. Consider a network input $a_j(t)$ to some unit j at time t. In general, the gradient is defined as

$$\delta_j(t) = \frac{\delta E}{\delta a_j(t)}, \qquad (7)$$

where E is the objective (error) function to be minimized and used for training. For a detailed explanation of the backward pass equations for each unit type cf. [20]. Since we are dealing with a regression problem, we consider the sum of squares error as a performance measure. The error function is defined as

$$E_t = \frac{1}{2K}\sum_i (y_i - y_i')^2, \qquad (8)$$

where K is a normalization factor which depends on the size of each sequence n_i and the total number of sequences in the dataset k. y_i is the output unit activation and y_i' is the expected value.

In order to test the convergence of LSTMs we used 10-fold crossvalidation sets for three different dataset sizes, namely 100, 200 and 500. That allowed us to estimate the approximate number of samples that are needed to learn the prediction task with high precision. In Fig. 13 a comparison of the average sum of squares error (SSE) and maximal and minimal SSE values is shown. In this case, the SSE is averaged among all the crossvalidation sets. The picture shows that the SSE and its standard

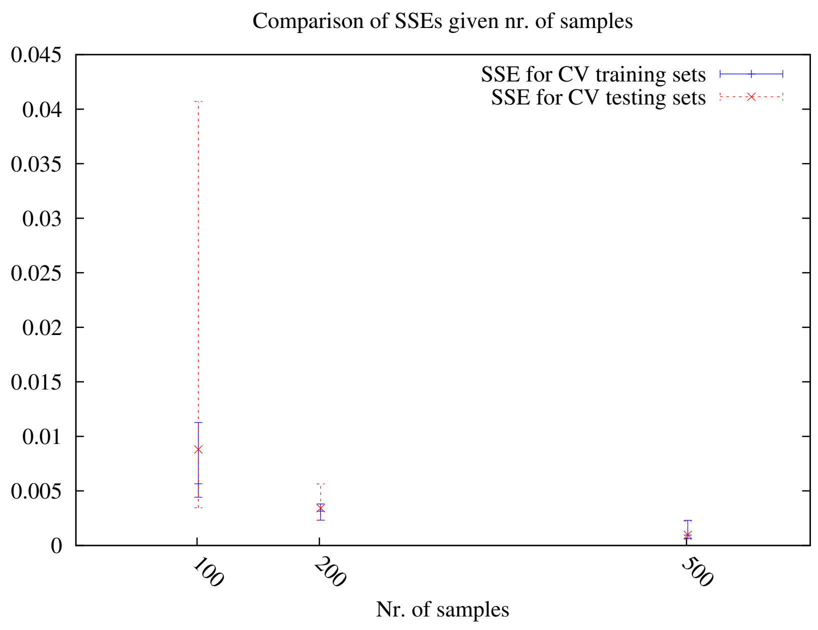


Figure 13: SSEs are reduced when increasing the dataset size.

deviation is considerably reduced when more samples are used, as expected.

LSTMs converge satisfactorily for these prediction tasks but we opted for a strategy based on obtaining qualitative models, which we describe in the following chapters.

3 LEARNING FEATURES FOR A PUSHING SCENARIO

This chapter describes the features our learning algorithms deal with. In this dissertation, we considered two different learning scenarios, the first one in simulation and the second one in a real tabletop environment. In the simulated scenario, the arm interacts with a polyflap, which is a polygon (concave or convex) cut out of a flat sheet of some material (e.g. cardboard) and folded once (anywhere) to produce a 3-D object [60], cf. Fig. 14. For both scenarios, we simulate a pushing action by applying a linear trajectory over a specified time period until the finger reaches a desired pose. The learning scenario in simulation is shown in Fig. 15. The simulated and real arms correspond to a Neuronics® Katana 6M^{TM}. In the simulated arm a sphere acts as a simple finger. The scenario in the real environment is shown in Fig. 16. The arm has 6 joints, including the last joint for the finger which is static. The representation of object poses is given in Euler angles with respect to a reference frame which is the origin in the scene (6-D pose).

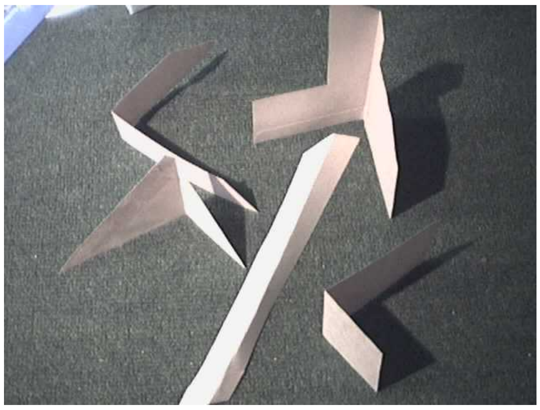


Figure 14: Polyflaps, `http://www.cs.bham.ac.uk/~axs/polyflaps/`. Here we used polyflaps of the shape shown in the bottom-right corner.

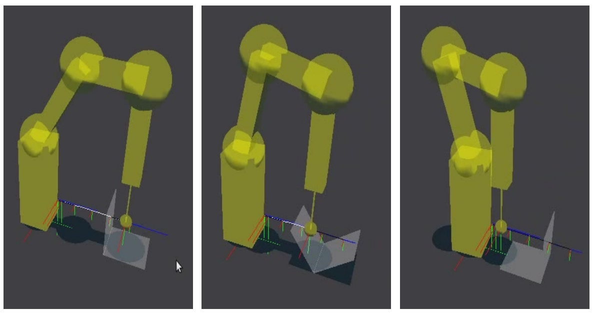


Figure 15: Learning scenario with a polyflap.

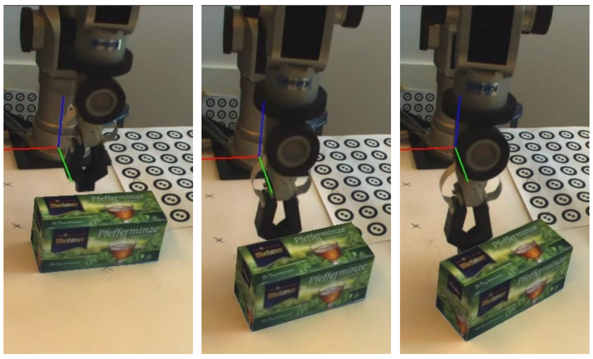


Figure 16: Learning scenario with a box.

For the estimation of poses in the real environment we use a tracker system based on sequential important resampling (SIR) particle filters [41]. This is a model-based tracking system using color and edge information from shape and texture. In Fig. 16 the object (tea box) shown in the image is actually a textured object model during the tracking process. The tracker is also used for generating the ground truth for evaluating the prediction experiments. The tracking requirements are real-time performance, i.e. within the frame rate of a typical camera (25-50 Hz), robustness to different lighting conditions, partial occlusion and motion blur. To achieve this, the approach is based on the following methods [41]:

- *Tracking-State-Detection (TSD)*: a TSD method was implemented to know whether the objects are tracked correctly, whether the object is occluded or whether it is lost. The knowledge of the tracking state (speed and confidence), allows for triggering online learning or pose recovery (see Fig. 17).

- *Texture mapping*: if texture is available, which is the case for our experiments, it is used for boosting robustness.
- *Pose recovery*: to initialise tracking and recover lost tracks distinctive features placed on the object's surface are used.
- *Online learning*: The feature points and texture of the object are automatically learned while tracking. In Fig. 18 we show an example of texture learning.

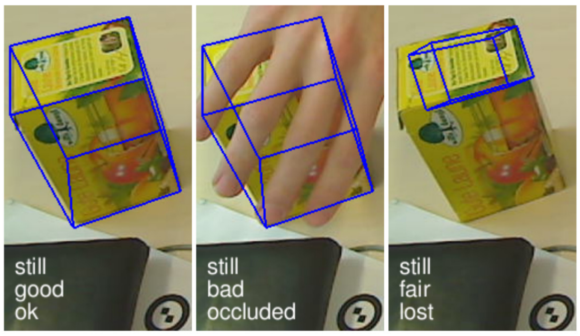


Figure 17: Tracking-State-Detection: From left to right: ok, occluded and lost tracking. [41]

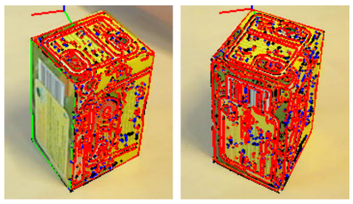


Figure 18: Succesively learning the texture of an object (red: matching edges from a textured face, green: matching edges from a non-textured face). [41]

[30, 31] provide a complete description of the rigid body transformations involved in a pushing scenario, assuming that the physical properties and net forces are constant in time. In summary, if two rigid bodies are present, where a time frame A corresponds to one object, a frame B to a second one, and a frame O for some fixed environment, rigid body transformations T between these frames at subsequent time steps can be used for describing the current system state. For instance, $T^{A(t),A(t+1)}$ represents the transformation of the first object from

time t to $t+1$. The prediction problem is stated as [30, 31]: given we know the starting states and the motion of the pusher $T^{A(t),A(t+1)}$, predict the resulting motion of the object $T^{B(t),B(t+1)}$.

In our work, we assume rigid body transformations with respect to a fixed frame O, although alternative representations are also possible. We assume a quasi-static environment, where frame velocities are not considered [29]. As in [29], causal relationships between the finger movement and the object movement are implicitly encoded in the representation, as we will clarify in Defs. 6, 7, 8, and 9. Rigid body transformations are encoded as rotation matrices which represent poses and can be transformed to 6-D poses consisting of position and orientation in Euler angles with respect to O.

Definition 5 *Let us denote $T^{E,O}, T^{A(t),O}$ and $T^{B(t),O}$ variables representing rigid body transformations with respect to a fixed frame O, where $T^{E,O}$ denotes the target pose representing the final expected pose of the finger at the end of a linear trajectory (desired pose), $T^{A(t),O}$ represents the pose of the finger at time step t and $T^{B(t),O}$ the pose of the object at t. We represent poses by using a 6-D representation encoding 3-D position and 3-D orientation in Euler angles, so that we have the vectors $\mathbf{m}, \mathbf{a}(t), \mathbf{b}(t)$ to denote finger target pose, finger pose and object pose respectively. We call $\mathbf{m}$ a motor command. The values are all normalized to obtain vectors with mean 0 and standard deviation 0.81650.*

We want to model a situated discrete time dynamical system (SDTDS) representing the interaction among these objects in the scenario. In an SDTDS [25], an input space $\mathcal{I} \subseteq \mathbb{R}^{n_i}$, a state space $\mathcal{S} \subseteq \mathbb{R}^{n_s}$ and an output space $\mathcal{O} \subseteq \mathbb{R}^{n_o}$ are defined, where n_i, n_s, and n_o are the finite dimensionalities of the spaces. In an SDTDS, a transition function $\gamma : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{S} \times \mathcal{O}$ is also defined, which allows to model the state of the system and its output, given some input. In Chapter 5, we explain how probabilistic finite-state machines can be constructed from $\mathcal{I}, \mathcal{S}, \mathcal{O}$ and γ. Below, we explain the process of collecting data for quantization.

Definition 6 *An SDTDS transition event at a time t, $\omega(t)$, is a quintuple $\langle \mathbf{s}(t), \mathbf{i}(t), \mathbf{o}(t-1), \mathbf{o}(t), \mathbf{s}(t+1) \rangle \in \mathbb{R}^{n_s} \times \mathbb{R}^{n_i} \times \mathbb{R}^{n_o} \times$*

$\mathbb{R}^{n_o} \times \mathbb{R}^{n_s}$*, where* $\mathbf{s}(t+1)$ *is the state vector reached after the SDTDS received input* $\mathbf{i}(t)$ *while occupying state* $\mathbf{s}(t)$*, and* $\mathbf{o}(t)$ *is the output generated in the transition [25]. Here, in contrast to [25], we also take into account previous output states (e.g.* $\mathbf{o}(t-1)$*) to consider a longer history of past events, with the intention to avoid the strict Markov property.*

Def. 7 provides a definition of the input space suitable for the pushing scenario.

Definition 7 *We define the input space* $\mathcal{I}$ *as a sensorimotor space where a set of concatenated tuples of vectors* $\langle \mathbf{m}, \mathbf{a}, \mathbf{b} \rangle$ *represent motor commands, finger poses and object poses respectively. For a time step* t*,* $\mathbf{i}(t) = \langle \mathbf{m}, \mathbf{a}(t), \mathbf{b}(t-1) \rangle$*;* $\mathbf{b}(t-1)$ *denotes the pose of the object at the previous time step.*

In a similar way, we can define the state space as follows.

Definition 8 *A state space* $\mathcal{S}$ *is defined entirely as a set of vectors representing object poses. For a time step* t*,* $\mathbf{s}(t) = \mathbf{b}(t)$*.*

The purpose behind the use of the features presented above is to eliminate ambiguities when predicting object trajectories. Given enough evidence about the current situation, represented in terms of an input vector, a state can be reconstructed with higher precision. This evidence is basically represented in the input space by a sensorimotor state.

As mentioned in Chapter 1, we can define different output functions to quantize an output space. Considering the problem of object trajectory prediction, we can encode an output space as a set of object transformations representing the rotation and translation of the object. In Chapter 5 we will clarify why this representation is useful for the purpose of object motion prediction.

Definition 9 *Let us define an output space* $\mathcal{O}$ *as a set of transformation vectors representing object motions. For time step* t*,* $\mathbf{o}(t)$ *is calculated as follows. Let us define a transformation* $T^{B(t),B(t+1)}$ *to denote the rotation and translation of an object, with corresponding* *rotation matrix* $\mathbf{R}_{B(t),B(t+1)} = \mathbf{R}^{\top}_{B(t),O}\mathbf{R}_{B(t+1),O}$ *and translation vector* $\mathbf{p}_{B(t),B(t+1)} = \mathbf{p}_{B(t+1),O} - \mathbf{p}_{B(t),O}$. *Then,* $\mathbf{o}(t)$ *is the corresponding transformation vector describing translation and rotation in Euler angles.*

In Fig. 19, we show a diagram of the features involved in the learning process.

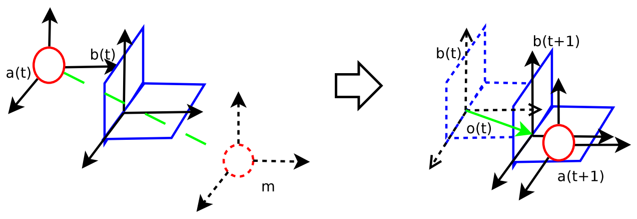


Figure 19: Features for prediction.

Defs. 7 and 9 define continuous vector spaces which will be discretized using a quantization algorithm. Alternatively, we can define an output space $\mathcal{O}$ as a set of symbols representing a class or type of rough object behavior. We carried out these experiments with simulated polyflaps. We consider here a discretization of possible object motions. For instance, $\mathcal{O} = \{-1, 1, 0, -0.5, 0.5\}$ is a set of possible values for an output symbol $o(t)$, denoting respectively:

- $o(t) = -1$ when the θ object angle (with respect to Z axis) decreases (this happens when the object tilts but does not completely flip over, so that it returns to the original angle).
- $o(t) = 1$ when the θ angle increases (object falling down)
- $o(t) = 0$ when the object does not move
- $o(t) = -0.5$ when the object moves backwards (negative direction along X axis).
- $o(t) = 0.5$ when the object moves forwards.

Additionally, we designed the output space to consider the final resulting motion, in abstract terms. In this case, we distinguished three possible behaviors, namely sliding, flipping over and tilting. Combined with the discretization presented above, we have 15 possible output symbols. By applying this discretization we can obtain predictions of motion categories in a similar way as in [51, 52].

4 ROBUST VECTOR QUANTIZATION

When dealing with learning problems, it is essential that information coming from sensors is compressed as much as possible. Learning tasks usually involve a high-dimensional set of vectors where information is often redundant and sparse. Thus, we need strategies for making generalizations about data, dimensionality reduction and efficient data representations. Compressing information is even more important when dealing with noisy data sets. As pointed out in [22], every regularity in the data can be exploited to compress information. One naive approach is the division of a space into regular cells [6], but with more variables (more dimensions) the number of cells grows exponentially. Moreover, an a priori division of the space might not capture the relationships between these variables well. In this chapter, we explain quantization algorithms used for an optimal subdivision of sensorimotor spaces. In Section 4.1 we describe in detail our quantization algorithm and in Section 4.2 quantization experiments with synthetic data sets are presented, which helps us to visualize optimal information compression with simple data distributions.

4.1 VECTOR QUANTIZATION

The ability to estimate a probability density function online is important for robots to learn in an incremental way, after new data become available for learning. Before we start explaining our method, it is important to distinguish the concepts of “online” and “incremental” in contrast to batch (Section 2.3.1). Here, by incremental we mean that during a learning iteration where a new data source is present, the learning algorithm can be updated immediately. However, a future improvement of our algorithm is the implementation of an online method in the strict sense, namely that a previously used data set is not anymore available for learning but only the new data items.

Our approach for density estimation is based on Vector Quantization [19] to partition probability distributions. In this way, it can model probability density functions. A related field of research is cluster analysis [26]. The idea is to map a set of high-dimensional vectors to a prototype vector, thereby identifying clusters in the data set that are statistically similar.

Following the machine learning terminology, a variable that represents a cluster is defined as a latent variable [6]. This variable defines assignments of data points to specific components of a mixture of distributions [6]. Technically, we have a data set $\mathcal{D}$ of points $\mathbf{x}_k \in \mathbb{R}^d$ where $d \in \mathbb{N}$ is the space dimensionality, a set of subsets or distributions $\{S_i \subset \mathcal{D}\}$, a prototype vector $\mathbf{x}_i$ that represents the cluster S_i, and the latent variable or index i. Moreover, some distance function or metric δ is used to evaluate the similarities among points in the data set.

Definition 10 *A quantizer* Λ *is a quadruple* $\langle C, S, \delta, \gamma \rangle$*, where* C *is a set of prototype vectors,* $S \subseteq \mathcal{D}$ *is a finite collection of* M *clusters,* δ *is a distance or metric function and* $\gamma : S \to \{1, 2, \dots, M\}$ *is a function that maps an element* $\mathbf{x} \in S_i \subset S$ *to its corresponding prototype index* i*.*

To find proper quantizations, a learning algorithm has to be implemented. The work presented here builds upon previous implementations of an incremental algorithm called Growing Neural Gas (GNG) [16], which is based on a graph whose nodes adapt to the topology of the probability distribution. The GNG algorithm was discussed in Section 2.4.3.

Definition 11 *A Growing Neural Gas (GNG) network [16] is a graph* $\mathcal{G} = \langle A, \mathcal{C} \rangle$ *where* A *is a set of* M *(cf. Def. 10) nodes, in which each node* $c \in A$ *has an associated weight* $\mathbf{w}_c \in \mathbb{R}^d$*. There exists a set of neighborhood connections* $\mathcal{C}$ *which are unweighted and symmetric. For each node* c*, there exists a possibly empty set of neighborhood connections* $N_c = \{i \in A \mid (c, i) \in \mathcal{C}\}$*. A GNG network can be used as a learning algorithm for a quantizer, by using the weights and indices associated to nodes in* A *as the prototype vectors and indices respectively.*

The algorithm has several strengths. It is an incremental algorithm which starts with 2 nodes and gradually adds more nodes in regions where a global error measure is high. By using Hebbian learning [16], a winner node and its neighbors gradually move towards regions with higher errors. In this way, the quantization error is minimized and the prototype nodes preserve the topology [16]. However, usually a proper quantization needs to be found so that clusters in the data set are represented by one or at least only a few nodes. For cluster analysis, this algorithm has several issues, some of them described in [49, 50]. For instance, the method suffers from the overfitting problem given the fact that one has to always define the maximum number of nodes M needed for representation. Another important issue is the presence of noise in the form of outliers. In [49, 50], some techniques were implemented to improve the convergence of the algorithm for efficient cluster analysis and in the presence of noisy data, as we already pointed out in Section 2.4.3.

In the algorithms presented in [49, 50] parameters like the maximum number of nodes and the maximum number of iterations for a training epoch are still used. In this dissertation, we incorporated modifications in learning rates calculation less dependent on a priori parameters, insertion criteria for nodes, efficient implementation of nodes deletion, and MDL-based and network stability stopping criteria. For the purpose of incremental estimation of a probability density function, we are interested in updating the quantizer after a new training instance is available. These training instances are actually sequences of vectors containing sensorimotor information. The stopping decision based on MDL and graph stability is here especially useful.

In the following sections, we describe and explain the calculation of the different parameters involved in the learning process: the learning rates, the weight updates, the node insertion criterion and the minimum description length calculation. Finally, we present the whole learning algorithm, experimental results and evaluation.

4.1.1 *Learning Parameters*

4.1.1.1 *Learning rates and weights update*

In [16], GNG employs fixed learning rates for a winner node s_1 and its neighbors. To balance the contribution of each of them, in [49, 50] learning rates are monotonically decreased taking into account the current iteration in a learning epoch, initial and final expected learning rate values and a predefined maximum number of nodes, as we noted in Section 2.4.3. To avoid relying on these predefined a priori parameters, we propose a new method for obtaining the rates which is dependent on an instantaneous calculation of mean error change ratio that offers an alternative solution for decreasing the learning rates.

To obtain an error change ratio, we compute an estimation of the current mean error and the previous mean error. A similar approach was followed in [23], but there it was used in the context of a supervised learning task in a non-stationary distribution where the error in prediction is calculated from the expected distance to a target vector. In that work, a ratio is obtained from a short-term average error and a long-term average error to obtain a quality measure for learning that is used for calculating the learning rate. Here, the error change ratio calculation is inspired by the work presented in [47], where an exploratory learning agent selects a learning sample via a learning progress measure derived from a decrease in the mean error rate in prediction.

We obtain a harmonic mean error rate. This way we obtain error measures that are not strongly influenced by outliers. Assuming a data point $\mathbf{x}(t)$ is presented to the network and the corresponding winner weight $\mathbf{w}_{s_1}(t)$ is activated, the inverse error $e_{s_1}(t)$ associated to the node s_1 is

$$e_{s_1}(t) = \|\mathbf{x}(t) - \mathbf{w}_{s_1}(t)\|^{-1}. \tag{9}$$

In this case, we assume $\|\cdot\|$ to be the Euclidean norm. Given a time window parameter τ during which a node s_1 was selected as winner node, a harmonic smoothed mean error rate for the node s_1 at the current time step t and at a previous time step $t-\tau$ is calculated as follows:

$$\begin{aligned}\langle e_{s_1}(t)\rangle &= \left(\frac{1}{\theta+1}\sum_{i=0}^{\theta} e_{s_1}(t-i)\right)^{-1} \\ \langle e_{s_1}(t-\tau)\rangle &= \left(\frac{1}{\theta+1}\sum_{i=0}^{\theta} e_{s_1}(t-i-\tau)\right)^{-1},\end{aligned} \tag{10}$$

where θ is a smoothing parameter.

For every node $k \in N_{s_1} \cup \{s_1\}$, the learning rate $\eta_k(t)$ that we propose is obtained in the following way:

$$\eta_k(t) = \begin{cases} \eta_k & \text{if } -(\log\langle e_k(t)\rangle - \log\langle e_k(t-\tau)\rangle) > 1 \\ -(\log\langle e_k(t)\rangle - \log\langle e_k(t-\tau)\rangle)\eta_k & \text{if } -(\log\langle e_k(t)\rangle - \log\langle e_k(t-\tau)\rangle) > 0.1 \\ 0.1\eta_k & \text{otherwise,} \end{cases} \tag{11}$$

whereby the default learning rate η_k is modulated by the error difference $-(\log\langle e_k(t)\rangle - \log\langle e_k(t-\tau)\rangle)$. We call this value a learning quality measure. This function is depicted in Figure 20 for $\eta_k = 0.5$ and serves to moderate the impact of the default learning rate for weights update. The thresholds in this function are selected by experimentation and should not be problem specific. Between 0.1 and 1 the error difference is not high, therefore we use these values to modulate the learning rate avoiding big changes in the weights update.

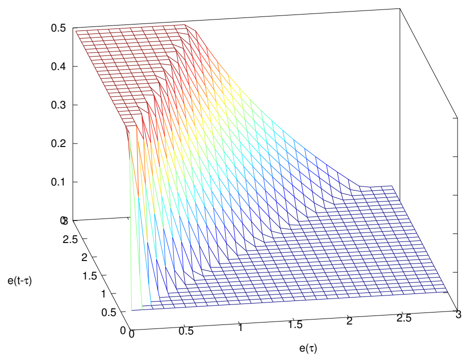


Figure 20: Learning quality function.

In the weight adaptation rule we also incorporated some strategies for outlier resistance based on the work described in [49]. The original formula in [16] is fragile to noisy environments and sensitive to the order of input vectors $\mathbf{x}$. Here we propose a new rule of the form:

$$\Delta \mathbf{w}_k(t) = \eta_k(t)\sigma_k(t)\frac{\mathbf{x}(t) - \mathbf{w}_k(t)}{\|\mathbf{x}(t) - \mathbf{w}_k(t)\|}, \tag{12}$$

where $\eta_k(t)$ is the learning rate in Eq. 11 and $\sigma_k(t)$ is the parameter for outlier resistance. From now on, we assume the time variable $t = 0$ at the beginning of a growth stage, that is, when a new node is added. The factor $\sigma_k(t)$ is used instead of the absolute distance information $\|\mathbf{x}(t) - \mathbf{w}_k(t)\|$, to mitigate the influence of outliers in the weight adaptation process. $\sigma_k(t)$ is calculated by using a historical restricting distance information $d_k(t)$ [49]:

$$\sigma_k(t) = \begin{cases} d_k(t) & \text{if } \|\mathbf{x}(t) - \mathbf{w}_k(t)\| \geqslant d_k(t-1) \\ \|\mathbf{x}(t) - \mathbf{w}_k(t)\| & \text{if } \|\mathbf{x}(t) - \mathbf{w}_k(t)\| < d_k(t-1) \end{cases} \tag{13}$$

where $d_k(t)$ serves as a restricting distance for $\mathbf{w}_k(t)$. $d_k(t)$ is updated when k becomes the winner node, i.e., when $k(t) = s_1(t)$:

$$d_k(t) = \begin{cases} \left(\frac{1}{2}\left(d_k(t-1)^{-1} + \|\mathbf{x}(t) - \mathbf{w}_k(t)\|^{-1}\right)\right)^{-1} & \text{if } \|\mathbf{x}(t) - \mathbf{w}_k(t)\| \geqslant d_k(t-1) \\ \frac{1}{2}\left(d_k(t-1) + \|\mathbf{x}(t) - \mathbf{w}_k(t)\|\right) & \text{if } \|\mathbf{x}(t) - \mathbf{w}_k(t)\| < d_k(t-1), \end{cases} \tag{14}$$

and

$$d_k(0) = \left(\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{w}_k^0\|^{-1}\right)^{-1}, \tag{15}$$

where $\mathbf{w}_k^0$ is the initial weight at the beginning of the growth stage and $N = |\mathcal{D}|$. Thus, $d_k(t)$ is initialized at the beginning and reinitialized when a new node is added to the network or when one or more nodes are deleted. We added a value $\varepsilon = 10^{-2}$ to Euclidean distances in order to avoid very big restricting distance values. The final updating rule is shown in Eq. 12, where ε is not shown for simplification.

4.1.1.2 *Criterion for node insertion*

Definition 12 *The node insertion requirement is fulfilled if for all nodes $c \in A$ the mean error rate e_c is not reduced after T_e learning epochs.*

A new node is added when the node insertion requirement is satisfied. The node is inserted in the proximity of a node q with maximal insertion criterion. An insertion criterion K_c is defined in our work simply as the highest mean error rate:

$$K_q(t) \quad = \quad \arg\max_{c \in A}(\langle e_c(t) \rangle) \tag{16}$$

In [16, 49, 50, 23] the weight calculation is based on an interpolation of the weight vectors of two nodes. The shortcoming of this method is that new nodes can not be added in proper locations due to the topological structure of some datasets. Therefore, in our method the location of the new prototype r is in fact calculated from the location of q and the direction of the mean average error vector $\langle \mathbf{e_q} \rangle$ associated to this node. This quantity is calculated in a similar way as in Eq. 10. When the data point $\mathbf{x}(t)$ is presented to the network, the mean error rate vector of the winner node s_1 is calculated as

$$\langle \mathbf{e}_{s_1}(t) \rangle = \left(\frac{1}{\theta + 1} \sum_{i=0}^{\theta} \mathbf{e}_{s_1}(t - i) \right)^{-1}, \tag{17}$$

with error vector $\mathbf{e}_{s_1}(t) = (\mathbf{x}(t) - \mathbf{w}_{s_1}(t))^{-1}$, where each component of $\mathbf{a}^{-1}$ is the inverse of the corresponding component of some vector $\mathbf{a}$. Thus, we set the weight for an inserted node r whose parent node is q as

$$\mathbf{w}_r(t) = \mathbf{w}_q(t) + 2\langle \mathbf{e}_q(t) \rangle. \tag{18}$$

The idea is to set the weight of the new node in the direction of the mean average error. It is multiplied by 2 to avoid weights which are too close each other.

In some cases, a dislocated node will be deleted according to a criterion explained in Section 4.1.1.4. Moreover, nodes are occasionally deleted at the end of the learning stage.

4.1.1.3 *Minimum Description Length Principle*

We use a Minimum Description Length criterion as proposed in [50] to determine the optimal number of clusters. It is an information-theoretic measure that balances the complexity of the graph and its error.

Definition 13 *Given a data set $\mathcal{D}$ and the set of prototype node weights $\mathcal{W}$, the MDL is defined as [50]:*

$$MDL(\mathcal{D}) = modelL(\mathcal{D}, \mathcal{W}) + errorL(\mathcal{D}, \mathcal{W}), \tag{19}$$

where $errorL(\mathcal{D}, \mathcal{W})$ is the total encoding length or model efficiency and $modelL(\mathcal{D}, \mathcal{W})$ is the model complexity.

The total encoding length and the model complexity for a network $\mathcal{G}$ with M nodes are calculated as follows:

$$\begin{aligned} \mathrm{errorL}(\mathcal{D}, \mathcal{W}) &= \kappa \sum_{i=1}^{M} \sum_{x \in S_i} \sum_{k=1}^{d} \max\left(\log_2\left(\frac{\|\mathbf{x}_k - \mathbf{w}_{ik}\|}{\varepsilon}\right), 1\right) \\ \mathrm{modelL}(\mathcal{D}, \mathcal{W}) &= KM + N \log_2 M, \end{aligned} \tag{20}$$

where $N = |\mathcal{D}|$, d is the dimension of input vectors, and ε is a data accuracy constant usually set to 10^{-4}. K is the number of bits needed to encode a single data vector, which is obtained according to the average value range ν and data accuracy ε: $K = \lceil \log_2\left(\frac{\nu}{\varepsilon}\right) \rceil$. Finally, κ is a parameter to balance the contribution of the model complexity and model efficiency, which is here usually set to 1. By setting $\kappa > 1$, we give more weight to the network accuracy in terms of error. The value range is calculated by obtaining the average value in the data set and substracting its lower limit.

4.1.1.4 *Criteria for node deletion*

In our work, we define two deletion criteria for nodes, thereby avoiding adding unnecessary complexity to the model:

1. A node is called inactive and can be deleted if at the end of a growth stage it is not a representative for any vector

in the dataset. Usually, new prototypes adapt quickly so that they approach regions containing data. However, in some cases clusters might be already represented by other prototypes and the node might stay isolated from clusters.

2. As in [50], we use the MDL criterion to assess whether a node is dislocated. Here, we apply the criterion at the end of the learning process. Assuming that one prototype f is removed, if the MDL value calculated based on the set of nodes $\mathcal{C} \setminus \{f\}$ is smaller than that of $\mathcal{C}$, i.e., if

$$\begin{aligned} \Delta\mathrm{MDL}(\mathcal{D}, f) &= -K + N(\log_2(M-1) - \log_2(M) \\ &+\kappa\Bigg(\sum_{i=1, i\neq f}^{M} \sum_{x\in S_i} \sum_{k=1}^{d} \max\left(\log_2\left(\frac{\|\mathbf{x}_k - \mathbf{w}_{ik}\|}{\varepsilon}\right), 1\right) \\ &- \sum_{i=1}^{M} \sum_{x\in S_i} \sum_{k=1}^{d} \max\left(\log_2\left(\frac{\|\mathbf{x}_k - \mathbf{w}_{ik}\|}{\varepsilon}\right), 1\right)\Bigg) < 0, \end{aligned} \tag{21}$$

we regard the removed prototype f as dislocated. In our algorithm, this process is performed recursively on the set of nodes until there are no more nodes dislocated.

4.1.1.5 *Robust GNG Algorithm*

We redesigned the learning algorithm in such a way that it can decide when to stop. This stopping criterion is based on evaluating the MDL after a number of T_m learning epochs after which no minimal graph has been found, in terms of MDL. The learning process is described in Algorithm 29.

Definition 14 *A graph is stable if the MDL criterion is not reduced after T_m learning epochs.*

In summary, the algorithm converges until the network is stable. We define convergence in terms of finding a graph which is optimal in the MDL sense. The calculation of MDL as explained in Section 4.1.1.3 is useful for evaluating graph stability, which implies that a good explanation of data, i.e. a good model, has been found. When the criterion of stability is not met, the solution is to add nodes in regions with higher mean error rates (as seen in Section 4.1.1.2). At the end of the learning phase, it is possible that some nodes will be dislocated. In that case, a more appropriate model with a smaller MDL must be found by deleting nodes (Section 4.1.1.4). Occasionally nodes are also removed when they do not become active till the end of a growth stage (Section 4.1.1.4).

Algorithm 2 : RobustGNG($\mathcal{D}, \mathcal{G}$)

```
Data : A data set $\mathcal{D}$ and a set of 2 non-connected nodes $\{c_1, c_2\} \in A$, whose weights are
       initialized randomly considering the data set bounds. Set the constants defined in
       Sections 4.1.1.1 and 4.1.1.3. Initialize restricting distances as explained in Eq. 15.
       Initialize smoothed mean error rates with the highest distance among the data set.
Result : A graph $\mathcal{G}$ which is stable
1  begin
2      while Graph $\mathcal{G}$ is not stable do
3          for $t = 1$ to $N$ do
4              Randomly draw a vector $\mathbf{x}(t) \in \mathcal{D}$;
5              Determine winner $s_1$ and second winner $s_2$, where
               $s_1 = \arg\min_{i \subset A} \|\mathbf{x}(t) - \mathbf{w}_i(t)\|$ and
               $s_2 = \arg\min_{i \subset A \setminus \{s_1\}} \|\mathbf{x}(t) - \mathbf{w}_i(t)\|$;
6              Update the weights of nodes $k \in N_{s_1} \cup \{s_1\}$ by using Eq. 12;
7              if $t \mod \lambda = 0$ where $\lambda$ is a time window constant then
8                  Calculate MDL using Eqs. 19,20;
9                  if current MDL is minimal then
10                     Store current graph $\mathcal{G}$ as the graph with minimal MDL $\mathcal{G}_{min}$;
11                 if current graph is stable then
12                     Store $\mathcal{G}_{min}$ as the resulting stable graph;
13                     Finish the algorithm here;
14                 if insertion requirement is fulfilled (Sec. 4.1.1.2) then
15                     Delete inactive nodes if necessary according to Section 4.1.1.4.
                       If a node is deleted, initialize restricting distances $d_k$ and
                       recalculate MDL. Check if MDL is minimal and store it
                       accordingly. Check if the graph is stable, in which case the
                       algorithm finishes;
16                     Determine nodes $q$ and $f$ with maximal insertion criterions $K_q$
                       and $K_f$ according to Eq. 16;
17                     Insert a new prototype $r$ and set its reference vector as in
                       Eq. 18. Set initial smoothed mean error for $r$, $q$, and $f$:
                       $e_r = e_q = e_f = 0$;
18                     Insert edges connecting the new prototype $r$ with prototypes $q$
                       and $f$, removing the original one: $\mathcal{C} = \mathcal{C} \cup \{(r, q), (r, f)\}$,
                       $\mathcal{C} = \mathcal{C} \setminus \{(q, f)\}$;
19                     Initialize restricting distances as explained in Eq. 15;
20             Update the smoothed harmonic mean error rates according to Eq. 10;
21             Update the restricting distances $d_k$ as in Eq. 14;
22             if a connection between $s_1$ and $s_2$ does not exist already and $s_1$ or $s_2$ has not
               been deleted then
23                 Create connection: $\mathcal{C} = \mathcal{C} \cup \{s_1, s_2\}$;
24                 Set the age of the connection $\{s_1, s_2\}$ to $0$: $age_{\{s_1,s_2\}} = 0$;
25                 Increment the age of all edges emanating from $s_1$:
                   $age_{\{s_1,i\}} = age_{\{s_1,i\}} + 1, \forall i \in N_{s_1}$;
26             Remove edges with age values greater than a constant $\alpha$;
27             Remove all nodes without any edge and in such case initialize restricting
               distances;
28     Check dislocated nodes according to 4.1.1.4;
29     return $\mathcal{G}$
```

4.1.2 *Active Learning*

After a new traning instance is available, RobustGNG is employed for quantization of input and output spaces. The action that a robot performs can be selected randomly or via an active learning procedure, which takes advantage of the information-theoretic model efficiency explained in Section 4.1.1.3.

Our method is based on previous work [47, 54] on Intrinsic Motivation Systems. The general idea of the Intelligent Adaptive Curiosity (IAC) algorithm used in these systems [47] is that a meta-learning system samples a set of actions and selects one that maximizes the learning progress. This is a measure based on the difference between smoothed current and previous mean error quantities.

In our work, we employ a slightly modified method where the selected action aims to minimize an information-theoretic measure of error. In that way, the robot will be intrinsically motivated to minimize the quantization error. However, it is important to remark that we are assuming here a learnable scenario, where the robot will not get stuck in unpredictable situations. Here, we want to evaluate the ability of our active learning algorithm to properly select actions aiming for minimizing a model error, in contrast to the work in [47] which investigates developmental phases in an almost uncontrolled scenario with different kinds of actions and objects. We used a near(ϵ)-greedy action selection rule with probability ϵ to allow random actions, thus permitting other exploration sources. Therefore, a random action is selected with a probability of ϵ which is usually set to 0.3.

Definition 15 *Let us define the normalized total encoding length or model efficiency for a data set $\mathcal{D}$ and the set of prototype node weights $\mathcal{W}$:*

$$normErrorL(\mathcal{D}, \mathcal{W}) = \frac{1}{N} errorL(\mathcal{D}, \mathcal{W}) \tag{22}$$

where $errorL(\mathcal{D}, \mathcal{W})$ is the total encoding length or model efficiency.

The current normalized model efficiency $\mathrm{normErrorL}_r$ is associated to the input quantizer $\Lambda_{r,i}$ of a region $\mathcal{R}_r$ in the sensorimotor space. This quantity will be used as the intrinsic reward for selecting an action in this region.

Definition 16 *Let $\mathcal{D}_r = \{S_i\}$ denote the set of instances (sequences) in region $\mathcal{R}_r$. Let us call a sequence tuple (instance) $S_i = \langle \mathbf{m}, \mathbf{a}_i, \mathbf{b}_i \rangle_{j=1}^{|S_i|}$ a tuple of tuples storing the motor command and finger and object poses obtained during one pushing action.*

Starting with one region, successive regions are obtained by splitting the sensorimotor space depending on a measure of variance in the data set $\mathcal{D}_r$ (exemplars used for Region $\mathcal{R}_r$). This space splitting makes also the learning process more efficient, since it is a divide-and-conquer strategy that reduces the computational complexity for processing large amounts of data. The division is performed after $|\mathcal{D}_r|$ achieves a certain threshold ϕ. A dataset $\mathcal{D}_r$ for a Region R_r is split in two datasets $\mathcal{D}_{r+1}, \mathcal{D}_{r+2}$ (for regions $\mathcal{R}_{r+1}, \mathcal{R}_{r+2}$). Then the split of $\mathcal{D}_r$ defined by the index c with value v_c is performed when the following criterion (Γ) is met:

- all the instances S_i of $\mathcal{D}_{r+1}$ have the cth component of their motor command vector $\mathbf{m}_i$ smaller than v_c.
- all the instances S_i of $\mathcal{D}_{r+2}$ have the cth component of their motor command vector $\mathbf{m}_i$ greater than v_c.
- the quantity $|\mathcal{D}_{r+1}| \cdot \sigma(\{[\mathbf{a}_{ij}\ \mathbf{b}_{ij}]_{j=1}^{|S_i|} \in \mathcal{D}_{r+1}\}) +$ $|\mathcal{D}_{r+2}| \cdot \sigma(\{[\mathbf{a}_{ij}\ \mathbf{b}_{ij}]_{j=1}^{|S_i|} \in \mathcal{D}_{r+2}\})$ is minimal, where

  $$\sigma(\mathcal{S}) = \frac{\sum_{v \in \mathcal{S}} \|v - \frac{\sum_{v \in \mathcal{S}} v}{|\mathcal{S}|}\|^2}{|\mathcal{S}|},$$

  and $\mathcal{S}$ is a set of vectors.

Each region stores all cutting dimensions and values that were used in its generation as well as in the generation of its parent regions. For the region $\mathcal{R}_r$ input and output quantizers $\Lambda_{r,i/o}$ are stored, and these machines are inherited by the child regions. The learning process is described in the Algorithm 3 for

I iterations (learning samples).

Algorithm 3 : Active learning process

```
Data : An initial region $\mathcal{R}_0$ which encompasses the whole sensorimotor space.
Result : A set of regions $\{\mathcal{R}_r\}$ with corresponding input and output quantizers $\{\Lambda_{r,i/o}\}$.
1 for i=1 to I do
2     Choose a motor command action $\mathbf{m}_{r,i} = \arg\max_{\mathbf{m}\in\{R_r\}}\{\text{normErrorL}_{r,i}\}$ among a set of 1000 candidate actions from all current regions $\{\mathcal{R}_r\}$ by using a near-to-greedy policy with probability $\epsilon$;
3     if $\phi$ is achieved then
4         Split region $\mathcal{R}_r$ into $\mathcal{R}_{r+1}$ and $\mathcal{R}_{r+2}$ according to $\Gamma$;
5         Recalculate dislocated nodes according to 4.1.1.4 for the regions $\mathcal{R}_{r+1}$ and $\mathcal{R}_{r+2}$;
6     end
7     Update quantizers with current training sequence $S_{r,i}$ and associated error $\text{normErrorL}_{r,i}$;
8 end
```

The splitting process is shown graphically in Figure 21.

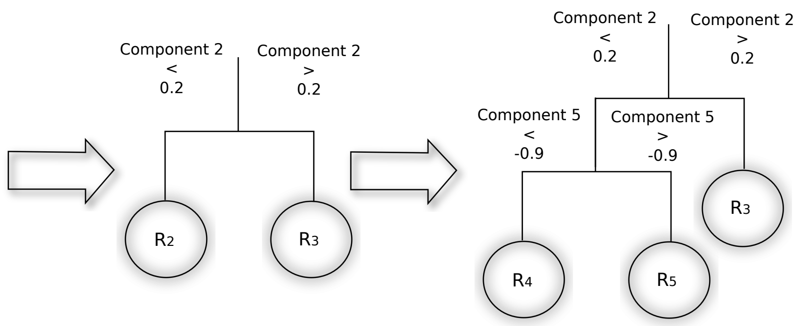
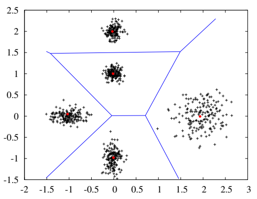
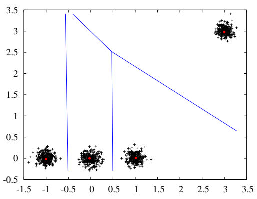


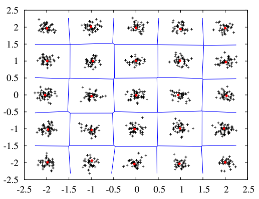
Figure 21: Region splitting process employing a criterion based on data set variance.

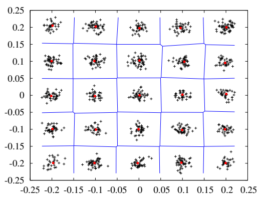
4.2 QUANTIZING SYNTHETIC DATA SETS

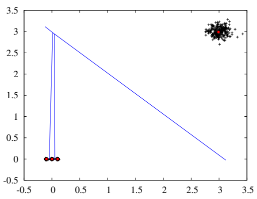
We designed several synthetic probability distributions in $\mathbb{R}^2$ with a variety of topological properties for testing our quantization method. We also used similar ones to those described in [49, 50]. We added white noise to test the algorithm in the presence of outliers. In Figure 22, different clean data sets are used to illustrate the results. The nodes are shown as red circles and the Voronoi regions they form are also illustrated. The variances of the Gaussian distributions are also variable, having values ranging from 0.01 to 0.3.

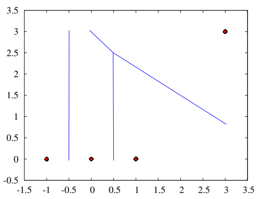

(a) Distribution 1

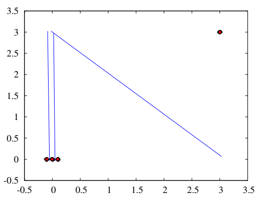

(b) Distribution 2

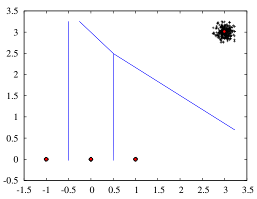

(c) Distribution 3


(d) Distribution 4


(e) Distribution 5


(f) Distribution 6


(g) Distribution 7


(h) Distribution 8

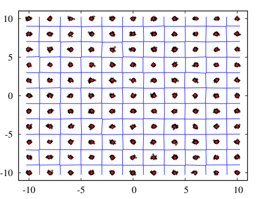
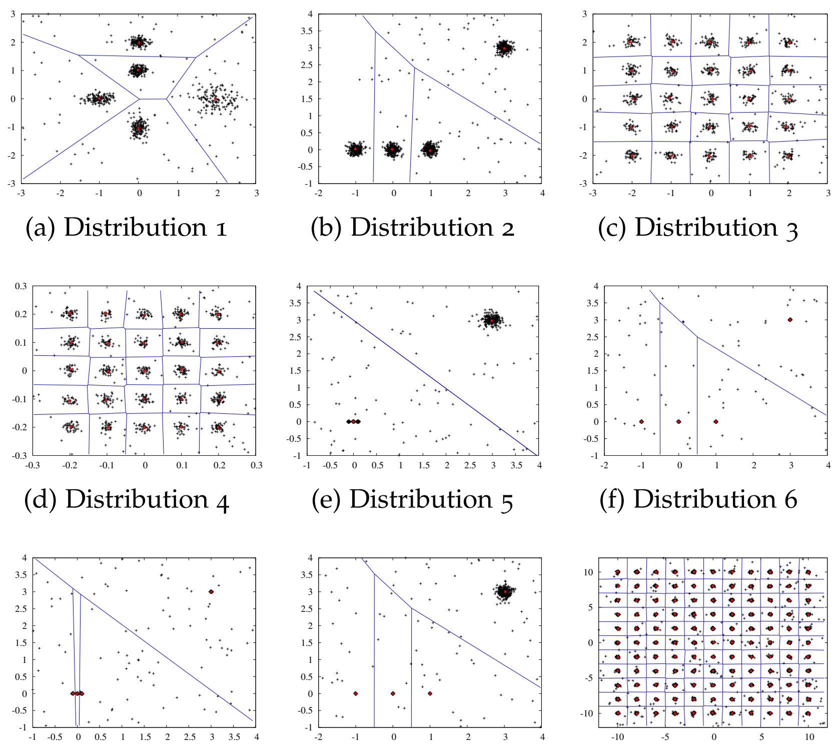

(i) Distribution 9

Figure 22: Synthetic data sets used for testing vector quantization.

The distribution 1 (Fig. 22a) consists of 5 Gaussian distributions centered at $[0, 2], [0, 1], [2, 0], [-1, 0]$ and $[0, -1]$ with variances of $[0.1, 0.1]$, $[0.1, 0.1]$, $[0.3, 0.3]$, $[0.2, 0.1]$ and $[0.1, 0.2]$ respectively. The distributions 2, 5, 6, 7 and 8 (Figs. 22b,22e, 22f, 22g and 22h) have two different variances ($[0.1, 0.1]$ and $[0.01, 0.01]$ respectively). The distributions 3 and 9 (Figs. 22c and 22i) have a variance of $[0.1, 0.1]$ for every component and the distribution 4 (Fig. 22d) a variance of $[0.01, 0.01]$.

We also generated data sets as described above, adding outliers generated by uniformly distributed white noise, as shown in Figure 23.



(a) Distribution 1 (b) Distribution 2 (c) Distribution 3

(d) Distribution 4 (e) Distribution 5 (f) Distribution 6

(g) Distribution 7 (h) Distribution 8 (i) Distribution 9

Figure 23: Synthetic data sets with outliers.

We ran 10 experiments for every distribution for evaluation. The parameters we used for all the experiments were $\eta_{s_1} = 0.3$, $\eta_{i \in N_{s_1}} = 0.001$, $T_e = 3$, $T_m = 200$, $\alpha = 50$, $\tau = 30$, $\theta = 50$. λ (the time window for insertions and deletions constant) usually equals the size of the data set, except for the distribution 9 where $\lambda = 1000$. For all distributions except 5 we used the parameter $\kappa = 1$. For the distribution 5 (Fig. 22e) we used $\kappa = 1.2$. As explained in Section 4.1.1.3, here we use $\kappa > 1$ to give more weight to the model efficiency, aiming for obtaining models that identify clusters with diverging variances.

In Table 2 we show the results of running the experiments on clean distributions. The column *Clusters* defines the number of *natural* clusters (ground truth). The column *No_Cluts* illustrates the average number of clusters that the algorithm finds during the 10 runs. The column *MSE* denotes the Mean Square Error and corresponding confidence interval for the 10 trials in each distribution. Here we assume a Gaussian distribution for MSE and No_Cluts values based on sample means and variances and a standard normal quantile $|z_{0.975}| = 1.96$. For every distribution we generated a distinct data set each time (generating 90 data sets in total). In this way, we deploy distributions with different characteristics to evaluate algorithmic robustness. In contrast, the algorithms RGNG (Robust Growing Neural Gas) and ENG (Enhanced Neural Gas) [49, 50] are only tested in 2 data sets for each synthetic distribution (6 data sets in total). In

Distribution	Samples	Clusters	No_Cluts	MSE
1	800	5	5 ± 0.0	0.034 ± 0.010
2	1000	4	4 ± 0.0	0.016 ± 0.002
3	1000	25	25 ± 0.0	0.028 ± 0.002
4	1000	25	25 ± 0.0	0.0028 ± 0.0001
5	1000	4	3.56 ± 0.55	0.022 ± 0.011
6	1000	4	4 ± 0.0	0.0015 ± 0.0002
7	1000	4	4 ± 0.0	0.0017 ± 0.0002
8	1000	4	4 ± 0.0	0.008 ± 0.004
9	5000	121	121 ± 0.0	0.026 ± 0.001

Table 2: RobustGNG results for clean synthetic data sets.

Table 3 we show the corresponding results for noisy data sets. Here an outlier is added to the data set with 9% probability, except for the distributions 9 (5% probability) and distribution 1 (10%). Choosing these parameters, the distributions 1, 3 and 9 are similar to the ones presented in [49, 50] which allows us to do some experimental comparisons. In Table 4 we compare the

Distribution	Samples	Clusters	No_Cluts	MSE
1	800	5	5 ± 0.0	0.037 ± 0.006
2	1000	4	4 ± 0.0	0.020 ± 0.004
3	1000	25	25 ± 0.0	0.030 ± 0.002
4	1000	25	24.89 ± 0.21	0.0036 ± 0.0007
5	1000	4	2 ± 0.0	0.059 ± 0.002
6	1000	4	4 ± 0.0	0.0022 ± 0.0005
7	1000	4	4 ± 0.0	0.0023 ± 0.0003
8	1000	4	4 ± 0.0	0.0049 ± 0.0014
9	5000	121	120.89 ± 0.21	0.03 ± 0.003

Table 3: RobustGNG results for noisy synthetic data sets.

distributions 3 and 9 with the corresponding ones in [49, 50]. However, due to lack of resources, we did not implement the RGNG and ENG algorithms described in [49, 50]. Thus, a fair comparison is not possible in this case, since it is not either possible to replicate the exact distributions in [49, 50]. Nevertheless, given the obvious similarities among these distributions, we present here comparable statistics to get an idea of the effectiveness of our procedure. We take the statistics for RGNG and ENG algorithms from [49, 50].

Distribution	Methods	Clusters	No_Cluts	MSE
Clean 3	RGNG	25	25 ± 0.0	0.003 ± 0.001
	ENG	n.a.		
	RobustGNG	25	25 ± 0.0	0.028 ± 0.002
Noisy 3	RGNG	25	25 ± 0.0	0.004 ± 0.001
	ENG	n.a.		
	RobustGNG	25	25 ± 0.0	0.030 ± 0.002
Clean 9	RGNG	n.a.		
	ENG	n.a.		
	RobustGNG	121	121 ± 0.0	0.026 ± 0.001
Noisy 9	RGNG	n.a.		
	ENG	121	121 ± 0.0	0.0024 ± 0.0001
	RobustGNG	121	120.89 ± 0.21	0.03 ± 0.003

Table 4: Comparison of different GNG-based algorithms on synthetic data sets.

These results show that we can achieve similar results with a method that assumes less parameters. For the noisy distribution 9, there was one case when RobustGNG only found 120 clusters. In that case, only one cluster was not identified after it was discarded by means of the MDL-based deletion mechanism. In that case, this cluster was considered noisy information, since there were not enough data items representing it and were considered outliers. This is illustrated in Fig. 24.

In Figure 25 we illustrate the evolution of MDL values. We ran an experiment with distribution 9 with both clean and noisy data sets. After the optimal number of clusters is found, MDL does not decrease and eventually, after T_m epochs, the algorithm stops. Finally, the graph with minimal MDL is stored, after checking possible dislocated nodes.

The quantization results show a good reliability of the algorithm, taking into account a variety of topological properties of

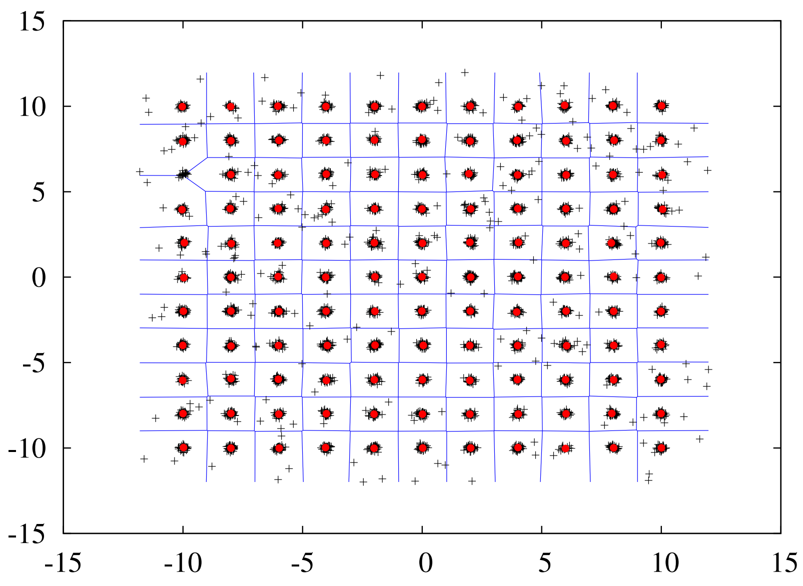
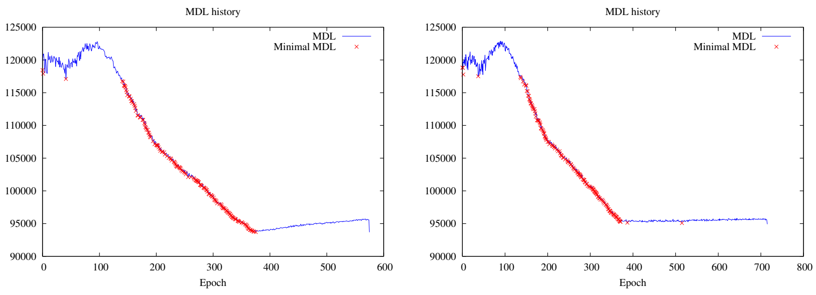


Figure 24: Suboptimal quantization for distribution 9.



(a) Clean data set (b) Noisy data set

Figure 25: MDL history for distribution 9.

the distributions and in the presence of outliers. The reduction of parameters for the algorithm is an additional gain, compared to previous GNG implementations. Likewise, an online calculation of the error and a learning rate based on this quantity makes the algorithm suitable to be subsequently implemented as an online learning algorithm (cf. discussion in Chapter 7). The modification of the weight calculation for a new inserted node is useful to find proper quantizations for the distributions 5, 6 and 8. The results also show that the procedure is invariant to different scale parameters (variances), so that it is not necessary to normalize data for the cases presented here. However, the algorithm is unable to identify clusters in the presence of noise for the distribution 5, due to the fact that one of the clusters has a high variance with respect to the others and the proximity of three clusters with a similar variance. Moreover, the insertion rule still favors the insertion of nodes with high average errors. A possible solution is to consider a density estimate of each distribution based on a frequentist approach to improve the insertion criterion.

5 INDUCTION OF SUBSTOCHASTIC SEQUENTIAL MACHINES

In this chapter, we summarize the methods that we apply for inducing sequential models of dynamical systems. In Section 5.1 we present the formalism of substochastic sequential machines, which are the models we infer from our learning scenarios. This formalism builds upon the quantization methods described already in Section 4.1. In Section 5.2 we describe the method for discretizing state spaces and in Section 5.3 the complete algorithm for extraction of models (Crystallizing Substochastic Sequential Machine Extractor - CrySSMEx).

5.1 SUBSTOCHASTIC SEQUENTIAL MACHINES

Definition 17 *A substochastic sequential machine (SSM) is a quadruple* $\langle Q, X, Y, \mathcal{P} = \{p(q_j, y_l | q_i, x_k)\}\rangle$ *where* Q *is a finite set of state elements (SEs),* X *is a finite set of input symbols,* Y *is a finite set of output symbols, and* $\mathcal{P}$ *is a finite set of conditional probabilities (cf. explanation in [25] and eqs.23-25) where* $q_i, q_j \in Q, x_k \in X$ *and* $y_l \in Y$.

We remarked in Section 2.5 that the original quantization mechanism in CrySSMEx is too simple to be applied to real noisy data sets. We use the vector quantization method described in Section 4.1 for discretization of input and output spaces and the CrySSMEx algorithm for the induction of SSMs which model dynamical systems. An SSM models a situated discrete time dynamical system (SDTDS). A stochastic dynamical model of such a system is a joint probability mass function p_Ω induced from a transition event set Ω, and quantizer functions Λ_o, Λ_i and Λ_s for output, input and state spaces respectively. Ω consists of selected transition events recorded from a given set of input sequences. Thus, the joint probabilities of observed and quantized transitions (p_Ω) are translated into joint probabilities of SSM transitions according to $\mathcal{P}$. As already mentioned, we define $\Lambda_i(\mathbf{i}(t))$ and $\Lambda_o(\mathbf{o}(t))$ according to the discretization described in Section 4.1, and $\Lambda_s(\mathbf{s}(t))$ using a modified version of the original state space quantization method (Crystalline Vector Quantizer - CVQ) explained in [25]. Thus, we have:

$$\begin{split} p(q_i, x_k, y_l, q_j) = \\ p_\Omega(\Lambda_s(\mathbf{s}(t)) = i, \Lambda_i(\mathbf{i}(t)) = k, \Lambda_o(\mathbf{o}(t)) = l, \Lambda_s(\mathbf{s}(t+1)) = j) \end{split} \tag{23}$$

The conditional probability is calculated with:

$$p(q_i, x_k) = \sum_{j=1}^{|Q|} \sum_{l=1}^{|Y|} p(q_i, x_k, y_l, q_j) \tag{24}$$

$$p(q_j, y_l | q_i, x_k) = \begin{cases} \frac{p(q_i, x_k, y_l, q_j)}{p(q_i, x_k)} & \text{if } p(q_i, x_k) > 0 \\ 0 & \text{if } p(q_i, x_k) = 0 \end{cases} \tag{25}$$

Definition 18 *The translation procedure from Ω to an SDTDS and then into an SSM will be called create_machine$(\Omega, \Lambda_s, \Lambda_i, \Lambda_o)$.*

The substochasticity of the extracted machines is due to the possibility that the sample of input sequences in Ω will not necessarily provide examples of all possible input symbols in all possible enumerations of the quantized space of the dynamical system. As a consequence, the probability distributions can become substochastic [25]. This, in turn, has the effect of deriving models which assume a closed world. Therefore, transitions which are not observed are not included in the model a priori.

The details of the procedure for extracting substochastic sequential machines are described in [25]. In summary, there is a recursive state splitting, starting from only one SE. Then, a decision to split data into different SEs is based primarily on the maximal output entropy

$$\arg\max H(Y|Q = q_i, X = x_k) = H(\mathcal{P}_y(q_i, x_k)), \tag{26}$$

and then on the maximal next state entropy

$$\arg\max H(Q|Q = q_i, X = x_k) = H(\mathcal{P}_q(q_i, x_k)). \tag{27}$$

This yields that state vectors that convey the most information (i.e., highly indeterministic) are used for splitting [25]. Here,

$$H(\mathcal{P}) = -\sum_{i=1}^{n} p_i \log p_i, \tag{28}$$

and

$$\begin{aligned} \mathbf{p}(q(t+1)) &= \mathcal{P}_q(q_i, x_k) \\ \mathbf{p}(y(t)) &= \mathcal{P}_y(q_i, x_k) \end{aligned} \tag{29}$$

are marginal distributions of $\mathcal{P}$. Each split node has associated model vectors that point to other split states, merged ones, or leaf nodes. The model vectors are calculated from the average of the vectors which they represent. Additionally, states are possibly merged if there exists an equivalence relation between two states based on determining when two SEs are not equivalent if they, in their outgoing transitions, share some input symbols and transitions that lead to discrepancies in the future output. The procedure finishes when the machine is deterministic, i.e., when the entropies for all states equal to 0. However, since it is improbable to converge completely to deterministic machines in a robotic learning scenario, we define in Section 5.3 a stopping criterion by setting a threshold on the number of iterations remaining until no improvement is achieved.

5.2 AN IMPROVED CVQ

As mentioned above, the quantization procedure for the state space is based on the CVQ quantizer. This method has similarities with hierarchical decision trees and learning vector quantization [25]. In principle, this is a supervised classification method. In this work, we redefined the quantization procedure of a CVQ with respect to [25], in order to solve issues regarding symmetrical properties present in the evolution of some dynamical systems (see Fig. 26 discussed below). A CVQ is defined as follows [25]:

Definition 19 *A CVQ Graph is a quadruple CVQ* $= \langle N_{\mathit{Leaf}}, N_{\mathit{VQ}}, N_{\mathit{Merged}}, n_{\mathit{root}} \rangle$ *where* n_{root} *is the root node of the CVQ graph and the constituents are defined below.*

Definition 20 *A leaf node* $n \in N_{\mathit{Leaf}}$ *has only one constituent,* $n = \langle i \rangle$*, where* $i \in \mathbb{N}$ *is an enumeration of the node within the CVQ and* $1 \leqslant i \leqslant |N_{\mathit{Leaf}}|$.

Definition 21 *A Vector Quantizer (VQ) node* $n \in N_{\mathit{VQ}}$ *is a tuple* $n = \langle \mathcal{M}, \mathcal{H} \rangle$ *where* $\mathcal{M}$ *is a list of* L *model vectors* $[\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_L]$*, where* $\mathbf{m}_i \in \mathbb{R}^d$ *and* $\mathcal{H}$ *is a nonrepetitive list of child nodes* $[h_1, h_2, \ldots, h_L]$ *where* $h_i \in N_{\mathrm{Leaf}} \cup N_{\mathrm{VQ}} \cup N_{\mathit{Merged}}$ *and* $d \in \mathbb{N}$ *is the dimensionality of the vector space.*

Definition 22 *A merged node in a CVQ graph,* $n \in N_{\mathit{Merged}}$*, contains only a "link",* $n = \langle n_{\mathit{group}} \rangle$*, where* $n_{\mathit{group}} \in N_{\mathit{Leaf}} \cup N_{\mathit{VQ}} \cup N_{\mathit{Merged}}$.

The model vectors in a VQ-node are associated with a list $[\ell_1, \ell_2, \ldots, \ell_L]$ of classifications (labels). An element ℓ_i is a tuple $\langle y(t_i), y(t_i + 1) \rangle$ which are the output symbols observed at some time step t_i and the subsequent $t_i + 1$. In this work, we redefine a CVQ quantizer function Λ_{cvq} in terms of a function winner : $N_{\mathrm{Leaf}} \cup N_{\mathrm{VQ}} \cup N_{\mathrm{Merged}} \times \mathbb{R}^d \times \langle \mathbb{N}, \mathbb{N} \rangle \to \{1, 2, \ldots, M\}$:

$$\Lambda_{\mathrm{cvq}}(\mathbf{s}(t)) = \mathrm{winner}(n_{\mathrm{root}}, \mathbf{s}(t), \langle y(t), y(t+1) \rangle), \tag{30}$$

which in turn is recursively defined as:

$$\mathrm{winner}(n, \mathbf{s}(t), \langle y(t), y(t+1) \rangle) = \begin{cases} \mathrm{ID} & \text{if } n \in N_{\mathrm{Leaf}} \\ \mathrm{winner}(n_{\mathrm{group}}, \mathbf{s}(t), \langle y(t), y(t+1) \rangle) & \\ & \text{if } n \in N_{\mathrm{Merged}} \\ \mathrm{winner}(h_w, \mathbf{s}(t), \langle y(t), y(t+1) \rangle) & \\ & \text{if } n \in N_{\mathrm{VQ}}, \end{cases} \tag{31}$$

where we determine w, the index of the winning child of a VQ-node, according to

$$w = \arg\min \|\mathbf{s}(t) - \mathbf{m}_i\|, \quad \text{s.t. } \ell_i = \langle y(t), y(t+1) \rangle. \tag{32}$$

The symmetry problem can be visualized in Fig. 26, where two model vectors with identical geometrical locations have different associated outputs. Thus, depending on the context, one of these model vectors has to be selected to eliminate the ambiguity. For instance, this problem may arise in our robotic learning scenario where we identify three subsequent states (defined by object poses) x_i, x_{i+1}, x_{i+2} for a situation when an object bounces. The object starts in state x_i, then goes to x_{i+1} and then returns to the original pose x_i. That is, $x_i = x_{i+2}$. In that case, two model vectors with the same value will have different outputs. Thus, by defining $\ell_i = \langle y(t), y(t+1) \rangle$ we withdraw the strict Markov assumption, because we define an output function which considers an additional time step.

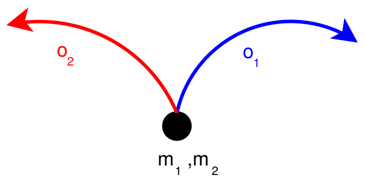


Figure 26: The symmetry issue in model vectors

We refer to [25] for more detailed explanations of CVQ training. When a CVQ leaf node is completely split [25], a recursive method is devised that splits data points in regions that separate them on the basis of labels for vectors as seen above. Model vectors are determined by averaging the data points in a region. Given the slightly different labeling used in this work, we reformulate the complete split in Def. 23. In Section 5.3 we describe the procedure for labeling vectors when splitting a data set.

Definition 23 *The complete split of several VQ nodes using several data sets at once is denoted* $\mathrm{cvq} = \textit{split\_cvq}(\mathrm{cvq}, D)$ *where* cvq *is the CVQ to be split and* $D = [D_1, D_2, \ldots, D_{|\Lambda_{cvq}|}]$ *is a list of data sets where* D_i *is the data set for splitting the leaf node with* $\mathrm{ID} = i$ *(if the node should not be split, then* $D_i = \emptyset$*). The elements of a data set are pairs* $\langle \mathbf{s}, \ell \rangle$ *where* $\mathbf{s}$ *is the data vector and* $\ell \in \langle \mathbb{N}, \mathbb{N} \rangle$ *is a label or class of the data vector. The leaf nodes are re-enumerated after the completion of all splits.*

In our work, most of the experiments were performed by applying a basic split [25] in order to avoid larger training times and, in our case, this decision apparently avoids less precise SSMs. The basic split does not involve the recursive splitting step in case that data vectors are not completely separated after the first split.

5.3 CRYSSMEX LEARNING LOOP

The principal components of the CrySSMEx algorithm are listed below [25]:

- the SDTDS, which represents the class of systems for CrySSMEx to analyze.
- the data set, i.e., the SDTDS transition event set Ω.
- SSMs, a subtype of SDTDSs.
- SDTDS transformation into SSM by quantizing input, output and state (cf. Sections 4.1 and 5.2).
- generation of UNDI-equivalence (universally not decisively inequivalent) sets in SSMs, which is the process that helps to determine when to merge states (function generate_UNDI_equivalence_sets).
- use of CVQ as a state space quantizer (cf. Section 5.2).
- merging (function merge_cvq) and splitting of CVQ leaf nodes (cf. Definition 23).
- selection and labeling of state vectors of Ω based on SSM information-theoretic properties (Algorithm 17).

Algorithm 4 : collect_split_data($\Omega, \Lambda_i, \Lambda_s, \Lambda_o$)

Data : A transition event set, Ω, an SSM, ssm, an input quantizer, Λ_i, a state quantizer, Λ_s and an output quantizer, Λ_o.

Result : A list of data sets D, one data set per $q \in Q$. An element of each data set is described in Def. 23.

```
begin
    $D = [\emptyset, \emptyset, \ldots, \emptyset]$;
    for $\forall \langle \mathbf{s}(t), \mathbf{i}(t), \mathbf{o}(t), \mathbf{s}(t+1) \rangle \in \Omega$ do
        $q_i = \Lambda_s(\mathbf{s}(t))$;
        $x_k = \Lambda_i(\mathbf{i}(t))$;
        $y_l = \Lambda_o(\mathbf{o}(t))$;
        $y_m = \Lambda_o(\mathbf{o}(t+1)$;
        $q_j = \Lambda_s(\mathbf{s}(t+1)$;
        if $\exists x_r : H_{ssm}(Y|Q = q_i, X = x_r) > 0$ then
            $x_{max} = \arg\max_{x_r \in X} H_{ssm}(Y|Q = q_i, X = x_r)$;
            if $x_k = x_{max}$ then
                $D_i = D_i \cup \langle \mathbf{s}(t), \langle y_l, y_m \rangle \rangle$;
        else if $\exists x_r : H_{ssm}(Q|Q = q_i, X = x_r) > 0$ then
            $x_{max} = \arg\max_{x_r \in X} H_{ssm}(Q|Q = q_i, X = x_r)$;
            if $x_k = x_{max}$ then
                $D_i = D_i \cup \langle \mathbf{s}(t), \langle y_l, y_m \rangle \rangle$;
    return $D$
```

In Algorithm 17, data sets are first split if output is indeterministic. If not, but the next state is not deterministic, data sets are then split and labeled accordingly. The diagram in Fig. 27 illustrates this process.

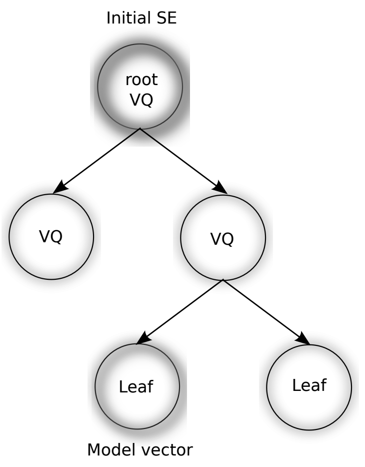


Figure 27: Partition of VQ states by the CVQ quantizer.

Finally, an improved version of the CrySSMEx main loop can be observed in Algorithm 16, where we added a stopping criterion.

Definition 24 *Let us define a constant τ needed to stop the algorithm when no improvement is observed after τ iterations. We set $\tau = 5$. In that case, the algorithm reaches a stability state.*

Algorithm 5 : CrySSMEx($\Omega, \Lambda_i, \Lambda_o$)

```
Data : An SDTDS transition event set, $\Omega$, an input space quantizer $\Lambda_i$,
       an output space quantizer $\Lambda_o$.
Result : A deterministic SSM mimicking the SDTDS within the domain
         $\Omega$ as described by $\Lambda_o$.
begin
    $i = 0$;
    $ssm^0 = create\_machine(\Omega, \Lambda_i, \Lambda_{cvq^0}, \Lambda_o)$ ($ssm^0$ has $Q = \{q_1\}$ with
    all transitions to itself);
    repeat
        $i = i + 1$;
        $D = collect\_split\_data(\Omega, ssm^{i-1}, \Lambda_i, \Lambda_{cvq^{i-1}}, \Lambda_o)$;
        $cvq^i = split\_cvq(cvq^{i-1}, D)$;
        $ssm^i = create\_machine(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$;
        if $ssm^i$ has UNDI-equivalent states then
            $E = generate\_UNDI\_equivalence\_sets(ssm^i)$;
            $cvq^i = merge\_cvq(cvq^i, E)$;
            $ssm^i = create\_machine(\Omega, \Lambda_i, \Lambda_{cvq^i}, \Lambda_o)$;
        if $ssm^i = ssm^{i-1}$ then
            Set $\iota = i$ the last iteration where improvement was found.
    until $ssm^i$ is deterministic or no improvement after $\tau = i - \iota$ iterations;
    return $ssm^i$
```

For every region $\mathcal{R}_r$, we executed the CrySSMEx algorithm, obtaining one SSM associated to some region.

An informative output function is essential for splitting the state space optimally. In our learning scenarios, object motion predictions and classifications benefit from the efficient state quantization which is obtained by applying the output functions described in Chapter 3. In the case of prediction, the approach was to use the quantized transformation vector as a way to eliminate ambiguities in next-state prediction. As for classification, the main purpose is to investigate the ability of the extracted SSMs to predict correct classifications and to analyze the state quantization which is produced. In the next chapter, we evaluate the prediction and classification ability of SSMs in an artificial dynamical system and in our robotic learning scenario, both of these having symmetrical properties and noisy state spaces.

6 EXPERIMENTAL RESULTS FOR PREDICTION AND CLASSIFICATION

As seen in Chapter 4, we developed RobustGNG to be integrated into our redesigned CrySSMEx algorithm (cf. Chapter 5). In this chapter, we use these learning methods to analyze the behavior of dynamical systems by inferring corresponding probabilistic models. In Section 6.1, we describe the analysis of the behavior of a noisy automaton and inference of the corresponding machine. This is useful for testing our algorithms with simple dynamical systems with symmetrical properties and noisy state spaces. In 6.2 we test and evaluate our algorithms on our robotic learning problem.

6.1 EXPERIMENTAL RESULTS FOR SSM INDUCTION WITH SYNTHETIC DATA SETS

To test the ability of the algorithm 16 to infer probabilistic machines from noisy data we carried out experiments with noisy automata. Experiments with noisy automata were proposed by [64]. Some GNG-based algorithms adapted to learn time series have already been used to solve this problem [1]. As seen in Section 2.5, state space models [6] and supervised spatiotemporal connectionist networks [33] have been applied for time series prediction. In previous work [64, 1], unsupervised algorithms have focused only on prediction, but not on models with optimal quantization and optimal rule construction from sequential data, as we do here. The goal is to evaluate the density estimating capabilities of a temporal model by reconstructing the transition probabilities of a second-order Markov model. Input vectors in $\mathbb{R}^2$ are generated from three Gaussian distributions with means $\mathbf{a} = [0, 0]$, $\mathbf{b} = [1, 0]$ and $\mathbf{c} = [0, 1]$ and common standard deviations σ. Figure 28 shows the automaton where transition probabilities are set depending on the parameter x.

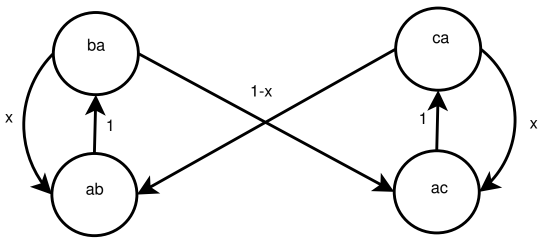


Figure 28: Transition probabilities of the noisy automaton.

We used the vector quantization algorithms to quantize the input spaces. In this case, the input space is derived from the probability distribution of the Gaussians. Thus, they generate three input clusters $\{a, b, c\}$ associated to means $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. For different values of σ and $x = 0.1$, Figure 29 shows the corresponding quantization for 1000 points.

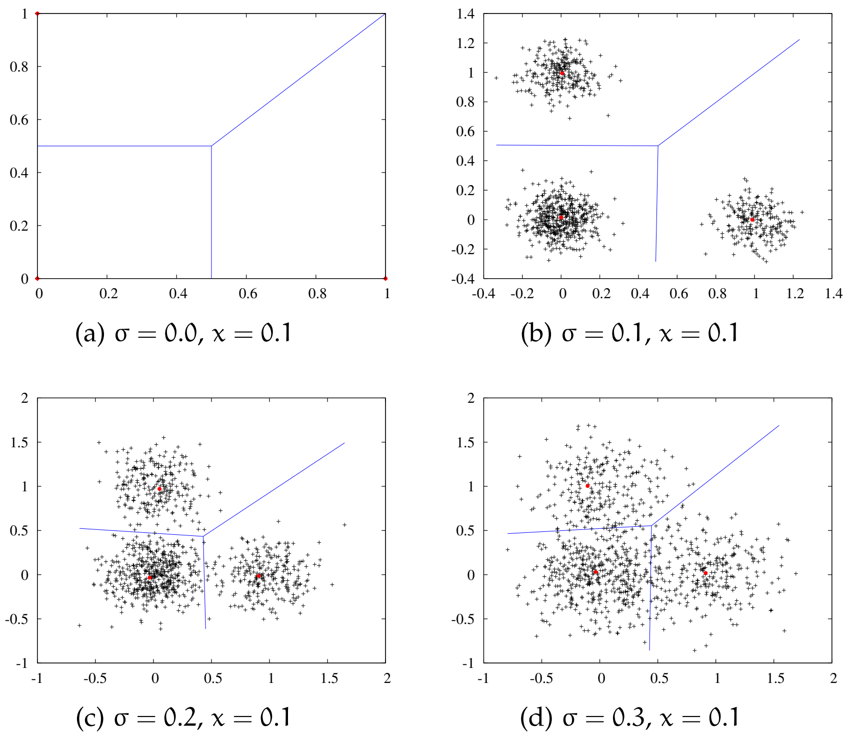
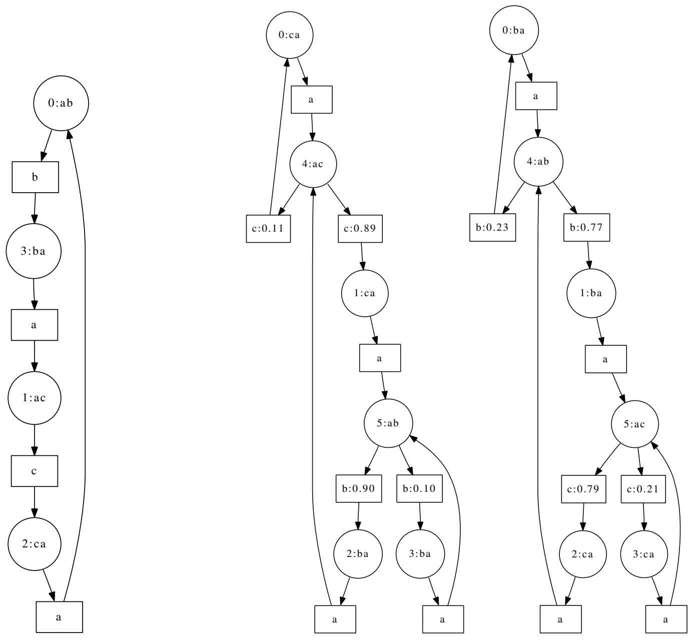

(a) $\sigma = 0.0$, $x = 0.1$
(b) $\sigma = 0.1$, $x = 0.1$
(c) $\sigma = 0.2$, $x = 0.1$
(d) $\sigma = 0.3$, $x = 0.1$

Figure 29: Input space quantization of noisy automata.

Here, we carried out experiments with 10^3 data points with standard deviation $\sigma \in \{0.0, 0.1, 0.2, 0.3\}$ and transition probabilities $x \in \{0.0, 0.1, 0.2, 0.3\}$. In this experiment, an output vector space is derived from the transition between a data point $\mathbf{i}(t-1)$ and $\mathbf{i}(t)$. We define an output vector $\mathbf{o}(t) = \mathbf{i}(t) - \mathbf{i}(t-1)$. Thus, the quantization algorithm RobustGNG is also used for discretization as the output quantizer Λ_o. In this case, it is easy to verify that the output space is discretized in 4 components.

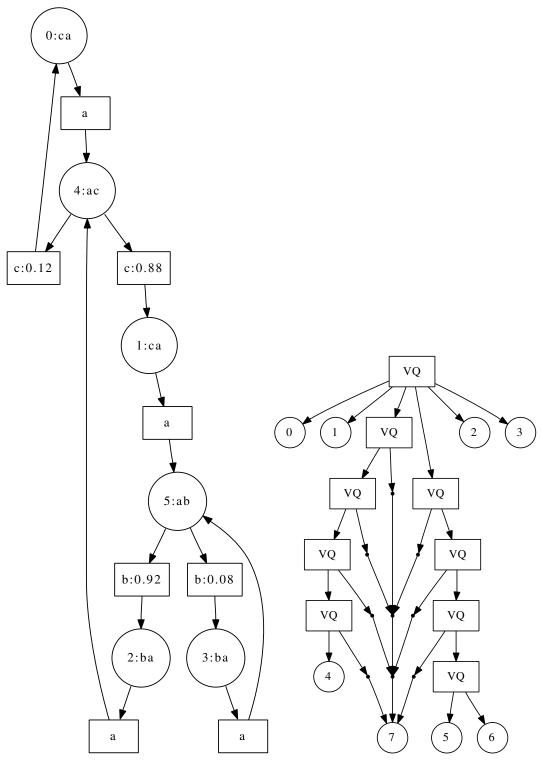
After the quantization of input and output spaces, we run the CrySSMEx algorithm. It is possible to induce either Moore-like SSMs or Mealy-like SSMs. Induced probabilistic Moore machines using the RobustGNG output quantization are visualized in Figure 30. The circles denote the states with corresponding output symbols. Here, an output symbol like ab denotes a transition where the input is a and next input is b. The boxes denote the transitions with corresponding input symbols and transition probabilities when applicable. For further splits the same transition probabilities are obtained but using more VQ-nodes. In Figure 31 the final CVQ tree for $\sigma = 0.0, x = 0.1$



(a) $\sigma = 0.1, x = 0.0$ (b) $\sigma = 0.1, x = 0.1$ (c) $\sigma = 0.1, x = 0.2$

Figure 30: Substochastic Moore Machines extracted from noisy automata

is visualized with the corresponding SSM, where boxes denote VQ nodes, points merge nodes and circles leaf nodes.



(a) SSM (b) CVQ Graph

Figure 31: CVQ Graph and corresponding SSM.

In Figure 30a, we observe that x was found to be 0.0 in both transition cases. In Fig. 30b, x is induced as 0.11 and 0.10, and in Fig. 30c to 0.23 and 0.21 respectively. Thus, transition probabilities were inferred with high accuracy. In Table 5 and 6 we present the reconstructed probabilities based on the induced probabilistic Moore machines. We show results for standard deviations $\sigma \in \{0.0, 0.1\}$. These results are similar to the ones shown by the Merge Growing Neural Gas (MGNG) algorithm, which was tested with $\sigma \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ [1]. We compare our method with MGNG in Table 6. Notice that the starting input symbol for all experiments was always a. For greater values of σ, the Gaussian distributions overlap, leading to more complex machines that are more difficult to analyze. In such cases, a point generated by some Gaussian distribution might be quantized as a member of other Gaussian distribution, since the margins that the Voronoi regions form are not sufficient to split points accordingly. In Fig. 32, the result for $\sigma = 0.2$ and $x = 0.0$ is illustrated.

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$P(a\|ba)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$P(b\|ba)$	0.0	0.08	0.2	0.32	0.42	0.51	0.63	0.68	0.8	0.91	1.0
$P(c\|ba)$	1.0	0.92	0.8	0.68	0.58	0.49	0.37	0.32	0.2	0.09	0.0
$P(a\|ca)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$P(b\|ca)$	1.0	0.88	0.8	0.68	0.58	0.52	0.39	0.29	0.18	0.12	0.0
$P(c\|ca)$	0.0	0.12	0.2	0.32	0.42	0.48	0.61	0.71	0.82	0.88	0.0

Table 5: Reconstructed probabilities for $\sigma = 0.0$.

CrySSMEx											
x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$P(a\|ba)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$P(b\|ba)$	0.0	0.1	0.23	0.29	0.39	0.5	0.59	0.68	0.85	0.89	1.0
$P(c\|ba)$	1.0	0.9	0.77	0.71	0.61	0.5	0.41	0.32	0.15	0.11	0.0
$P(a\|ca)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
$P(b\|ca)$	1.0	0.89	0.79	0.72	0.6	0.48	0.41	0.26	0.2	0.09	0.0
$P(c\|ca)$	0.0	0.11	0.21	0.28	0.4	0.52	0.59	0.74	0.8	0.91	0.0
MGNG											
x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$P(a\|ba)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	n.a.	n.a.
$P(b\|ba)$	0.0	0.098	0.201	0.302	0.398	0.498	0.603	0.699	0.796	n.a.	n.a
$P(c\|ba)$	1.0	0.901	0.798	0.697	0.601	0.501	0.396	0.3	0.203	n.a.	n.a.
$P(a\|ca)$	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	n.a.	n.a.
$P(b\|ca)$	1.0	0.9	0.798	0.7	0.6	0.498	0.398	0.3	0.2	n.a.	n.a.
$P(c\|ca)$	0.0	0.099	0.201	0.299	0.399	0.501	0.601	0.699	0.799	n.a.	n.a.

Table 6: Reconstructed probabilities for $\sigma = 0.1$.

In summary, in contrast to previous work, we are not only able to obtain probabilistic models which can predict well the transitions of a noisy automaton, but to infer their qualitative models by constructing an SSM version of this simple dynamical system via the unsupervised mechanism CrySSMEx. Without the redesign applied to CrySSMEx, this algorithm could not have been used for second-order Markov models with symmetrical properties. In the next section we show how our algorithm can be applied to the complex interactive dynamical systems involved in our learning robotic scenario, which also have the quality of being noisy and symmetrical.

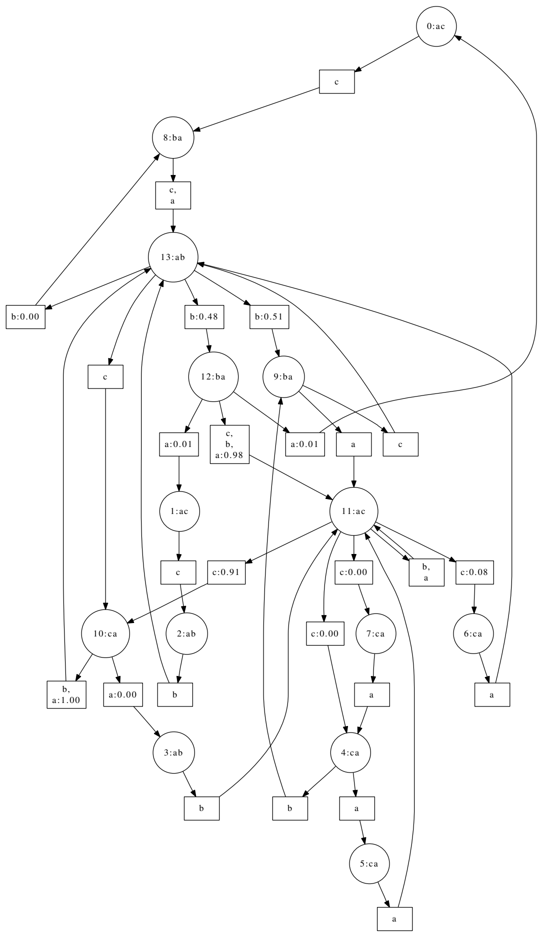


Figure 32: A complex automaton extracted from overlapping Gaussian distributions.

6.2 EXPERIMENTAL RESULTS FOR PREDICTION AND CLASSIFICATION

For evaluation purposes, we discuss two learning scenarios (real and simulated) where a robotic arm interacts with one object at the time. In the implementation we use the Golem library [29] which uses the Nvidia® PhysX^{TM} library enabling us to perform realistic physical simulations and to obtain feature vectors which are used for learning and evaluation. This learning setting can easily be re-adapted to real scenarios. Although they provide an idealized scenario, these experiments are necessary to establish a baseline from which we can start addressing noisy information in a real environment. In the simulated scenario the arm interacts with a polyflap, whereas in the real scenario, the arm interacts with a tea box, as shown in Fig. 7. The simulated and real arms correspond to a Neuronics® Katana 6M^{TM}.

6.2.1 *Feature selection assessment and learning parameters in the robotic scenario*

Chapter 5 refers to the automatic construction of SSMs which will be useful for prediction and eventually planning tasks. Section 6.1 showed that CrySSMEx can be applied to noisy data sets and symmetrical second-order Markov models, which is the case in the interactive robotic learning scenario. However, the conditional entropy evaluation which is linked to SSMs is also useful to evaluate the degree of uncertainty which will produce a certain selection of features for learning. After some initial tests and the evaluation of the conditional entropy associated to all possible combinations of states and input symbols in an SSM, a decision was taken favoring the feature selection presented in Chapter 3. These preliminary checkings triggered the decision of using Mealy machines instead of Moore machines. Conditional entropy values closer to 0 were preferred. However, an examination of the way a Mealy machine processes an input can shed light on the reason why Mealy machines are more efficient in this case.

In a Mealy machine, the output state is determined both by its current state and the current input, in contrast to the Moore machine, whose output is only determined by its current state. Thus, the more information we give about the current situation, in terms of input and state, the less ambiguous is the output value. Moreover, learning experiments such as the ones described in [47] involving a sensorimotor description of the state of the system use a similar configuration to define the features which serve as evidence in the current predicting situation.

In the learning scenario, the arm starts a pushing action from 18 different poses. The robot applies a pushing angle ranging from 60 to 120 degrees, parallel to the ground plane in the direction of the object center, as shown in Fig. 33, where the spheres show the starting poses of the finger. We set a constant speed for the robot movements.

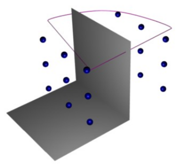
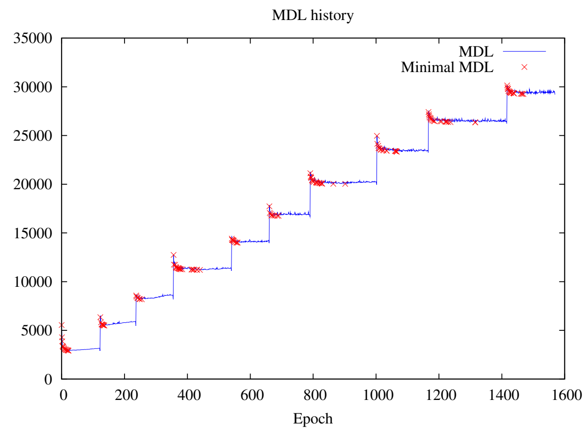


Figure 33: Pushing actions.

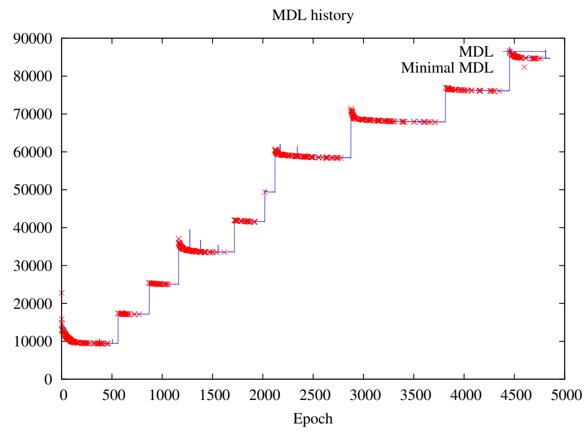
For input and output spaces quantized by means of Robust-GNG, we used the parameters $\eta_{s_1} = 0.8$, $\eta_{i \in N_{s_1}} = 0.001$, $T_e = 1$, $T_m = 100$, $\alpha = 50$, $\tau = 30$, $\theta = 50$, and λ equal to the size of the corresponding data set. For input spaces we used the parameters $\kappa = 1$ and $\varepsilon = 10^{-3}$ corresponding to the MDL criterion. For output spaces we used $\kappa = 10$ and $\varepsilon = 10^{-4}$. The difference in the last parameters lies in the fact that the scale of output vectors is very small. Thus, we adjusted these parameters to find more precise and informative quantizations. The splitting criterion ϕ is set to 10 iterations.

6.2.2 *Learning aspects*

We already defined in Defs. 7 and 9 the input and output spaces to be discretized and we described in Section 4.1 the discretization method. As we observed in 4.1.2, starting with one region, two associated quantizers are used for input and output space quantization, respectively, and each region $\mathcal{R}_r$ is potentially subdivided according to a variance measure in the dataset $\mathcal{D}_r$. In Figure 34 we illustrate the evolution of MDL values for a certain region. Initially, the algorithm runs with the first data set. After the optimal number of clusters is found, MDL does not decrease and eventually, after T_m epochs, the algorithm stops. Finally, the graph with minimal MDL is stored, after checking possibly dislocated nodes. When the data set is updated with new incoming data, we identify a new peak in the MDL and eventually it is again minimized.



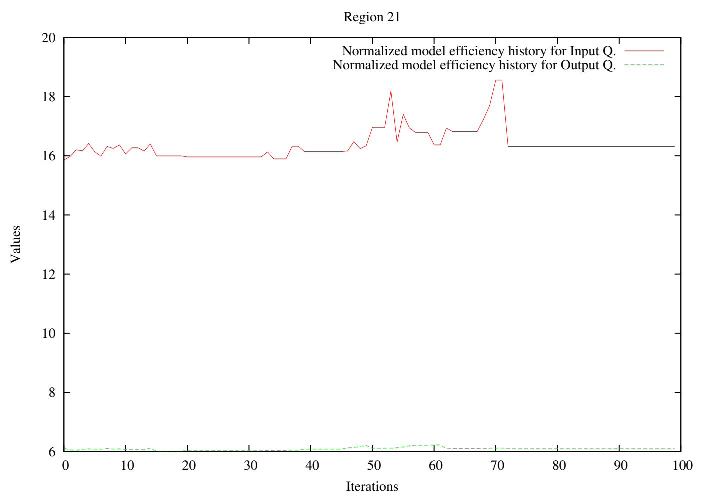
(a) Input quantizer



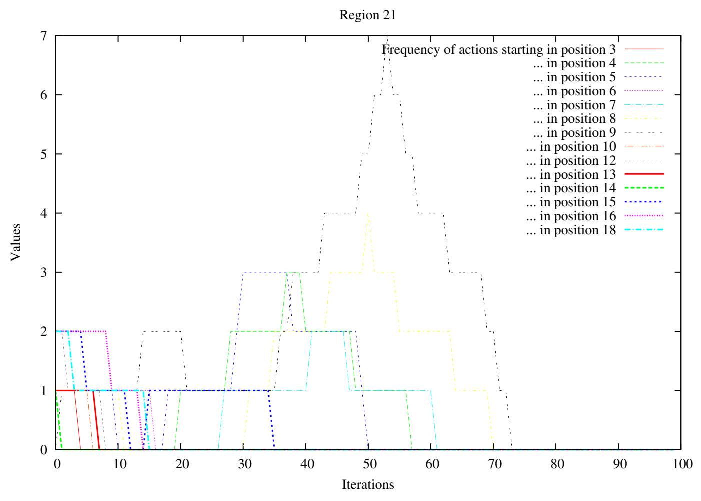
(b) Output quantizer

Figure 34: MDL history for a given sensorimotor region.

In the case of active quantization, when the normalized model efficiency normErrorL_r for an input quantizer in some region $\mathcal{R}_r$ becomes maximal, actions associated to this region are executed more frequently. Figure 35 illustrates the evolution of normErrorL_r and the frequency of actions for a region $\mathcal{R}_r$ by using a window of size 20 iterations. The actions are categorized according to one of the 18 possible starting positions of the pushing action. The figure shows the whole history of the regions, including the history inherited from parent regions since the creation of the root region.



(a) Normalized model efficiency



(b) Frequency of actions (window size: 20)

Figure 35: History of normalized model efficiency for the input quantizer of a given sensorimotor region.

Likewise, diagrams illustrating frequencies of actions as in Figure 36 give us information about the evolution of the active learning quantization procedure. The graphs show continuously changing peaks in the frequency of actions starting from certain poses, suggesting that the robot focuses on actions corresponding to regions where the normalized model efficiency is higher. Thus, the robot is inherently motivated to minimize such an error measure. However, the graph also shows that the type and the order of actions performed can vary for each experiment, which is natural given the stochastic nature of the greedy policy for selecting random actions. In contrast, a diagram for the online incremental learning case would obviously show uniform frequency values for all actions, as shown in Fig. 37.

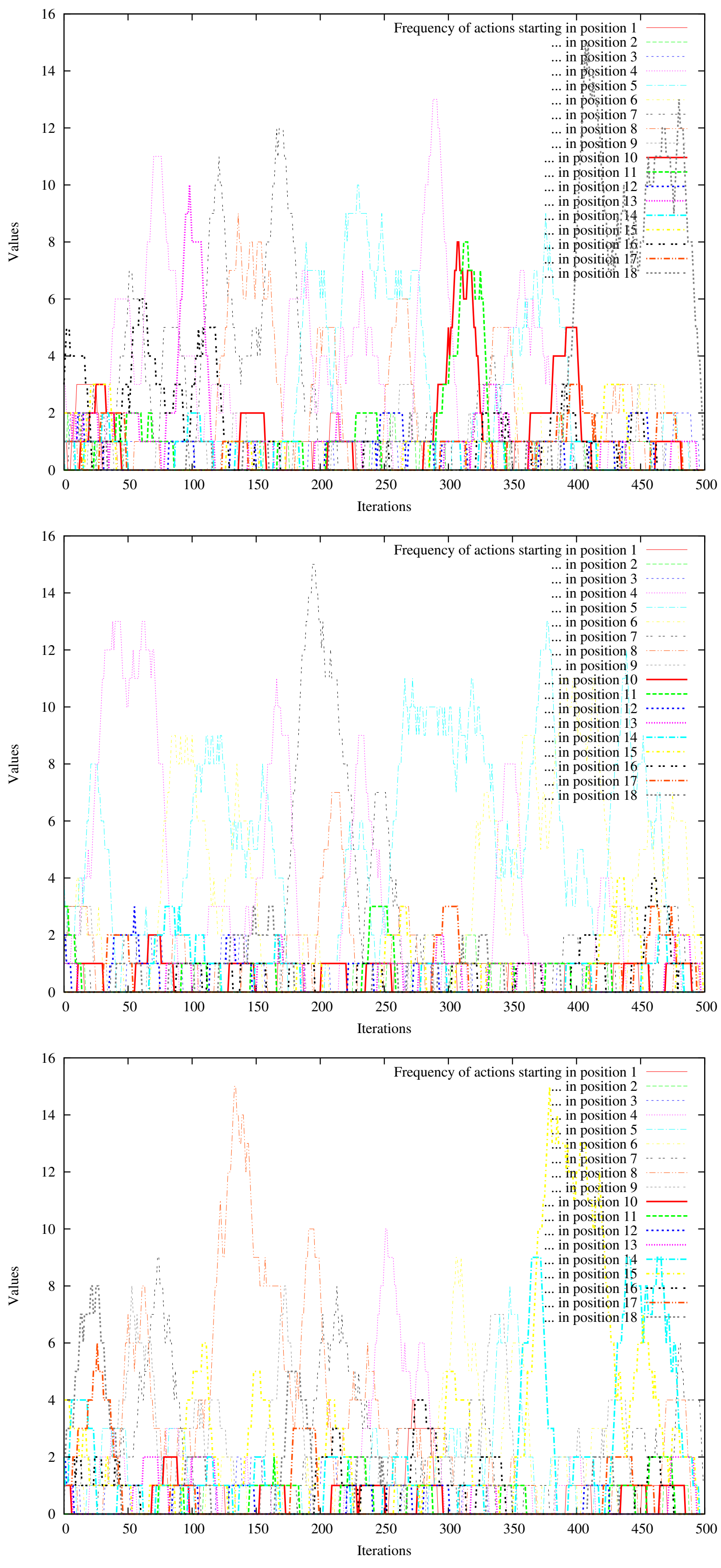


Figure 36: Three runs of the active learning quantization experiment.

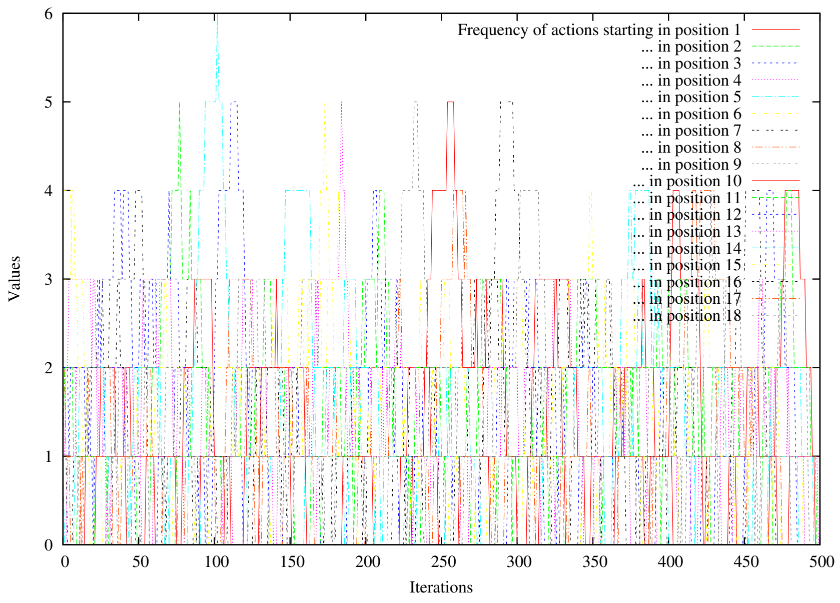


Figure 37: A run of the online quantization experiment.

6.2.3 *Prediction results*

The outcome of the quantization process is a set of quantizers $\Lambda_{r,i/o}$ for all regions. $\text{CrySSMEx}_r(\Omega_r, \Lambda_i, \Lambda_o)$ (algorithm 16) is run for every region $\mathcal{R}_r$ to obtain corresponding SSMs ssm_r. To test the prediction results, we applied the prediction process explained in [25], which is called a parsing process, because of its applicability to formal languages and automata. We performed experiments for both one-step ahead and long-term predictions.

6.2.3.1 *One-step ahead prediction*

Let $\mathbf{p}(q(t)) = [p(q_1(t)), p(q_2(t)), \ldots, p(q_n(t))]$ be a substochastic vector denoting the distribution over Q at time t and $x_k(t) \in X$ be the input symbol fed to the machine. The resulting distribution vector over Q, $\mathbf{p}(q(t+1))$, is calculated by [25]

$$\mathbf{p}(q(t+1)) = \mathcal{P}_q(\mathbf{p}(q(t)), x_k(t)) \tag{33}$$

where each element $p(q_j(t+1))$ of $\mathbf{p}(q(t+1))$ is calculated by [25]

$$p(q_j(t+1)) = \sum_{i=1}^{|Q|} \left(p(q_i(t)) \cdot \sum_{l=1}^{|Y|} p(q_j(t+1), y_l(t) | q_i(t), x_k(t)) \right) \tag{34}$$

These equations describe the prediction distribution of a state, given an input and a previous state. Likewise, the distribution of output symbols $\mathbf{p}(y(t))$ over Y is generated in the transition by [25]

$$\mathbf{p}(y(t)) = \mathcal{P}_y(\mathbf{p}(q(t)), x_k(t)), \tag{35}$$

where each element $p(y_l(t))$ of $\mathbf{p}(y(t))$ is calculated by [25][1]

$$p(y_l(t)) = \sum_{i=1}^{|Q|} \left(p(q_i(t)) \cdot \sum_{j=1}^{|Q|} p(q_j(t+1), y_l(t) | q_i(t), x_k(t)) \right) \tag{36}$$

In order to obtain the model vectors from the CVQ and calculate the normalized root mean square error in prediction, we obtained a map of model vectors and corresponding quantization indices to obtain an inverse quantization function Λ_s^{-1}. Since a VQ node in the CVQ quantizer may have more than one model vector, we obtained the map associating a VQ node index and a model vector by calculating the mean of its associated model vectors obtained using CrySSMEx.

We performed training experiments in simulation with 100, 200 and 500 sequences. In a first experiment, we applied an online incremental learning mechanism, by setting the greedy policy with probability $\epsilon = 1.0$, i.e., the robot only chooses random actions. In a second experiment, we applied the usual active learning procedure with $\epsilon = 0.3$. We repeated both experiments three times to obtain accurate statistics. The best results were obtained for 500 sequences and the online quantization procedure. Figure 38 shows the normalized root mean

1 We modified the notation used in [25] expecting to make the mathematical description more clear. For instance, an output $y_l(t)$ (instead of $y_l(t+1)$) is associated to the current input $x_k(t)$ and current state $q_i(t)$.

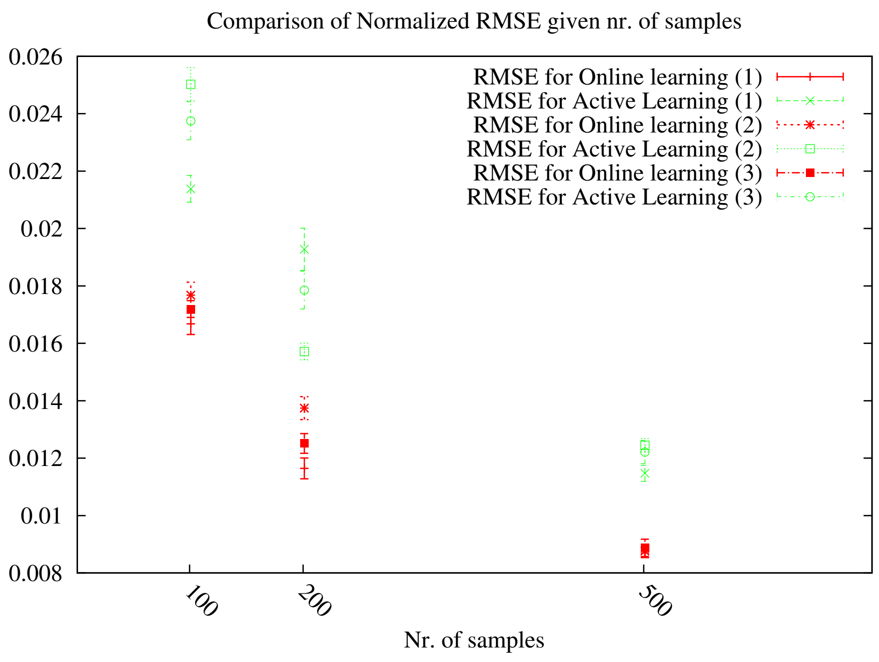


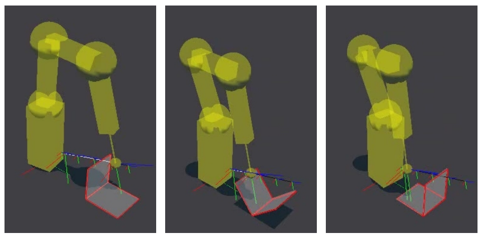
Figure 38: Normalized RMS error in prediction.

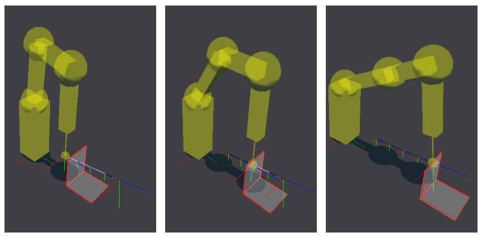
square error (RMSE) in prediction for these 3 cases, by using 10 different sets of 500 ground truth validation sequences. Here, the confidence interval for the mean of the RMSE is calculated from the sample mean and variance assuming a Gaussian distribution and a standard normal quantile $|z_{0.975}| = 1.96$. More iterations would give more accurate results, as the trend in the RMSE results shows. In Table 7 we show the average number of SSM states obtained for 3 online learning experiments and 3 active learning experiments, each for 3 different number of sequences.

Quantizers	Avg No. of States		
	100 sequences	200 sequences	500 sequences
Online	1697	3942	10873
Active	1731	3526	10748

Table 7: Average number of SSM states for 3 experimental cases in the simulated scenario for object trajectory prediction.

In Fig. 39 we exemplify the prediction results.


(a) Predicting the flipping over affordance


(b) Predicting the sliding affordance

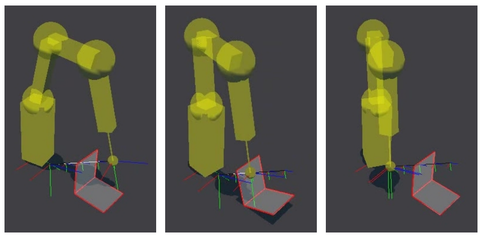

(c) Predicting the tilting affordance

Figure 39: Short-term prediction of affordances. Red contour depicts last prediction.

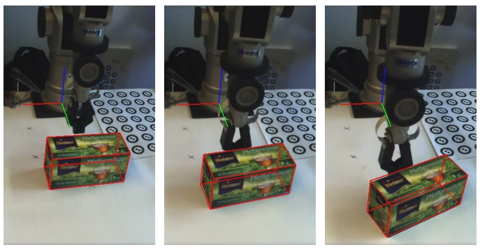
The algorithm was also tested within a real environment obtaining comparable results to the ones obtained in simulation. Here, we used the object tracking system to generate both learning and ground truth data, as described in Chapter 3. The learning scenario involves 120 pushing actions starting from one pose and applying a pushing angle ranging from 60 to 120 degrees parallel to the ground plane in the proximity of a tea box. We tested the online version of the algorithm, where the greedy policy is $\epsilon = 1.0$ as in the previous experiments. Likewise, we tested an offline version of the algorithm where there was no subdivision of the sensorimotor space in regions. Thus, each action was selected randomly and the sequences for learning the input and output spaces were stored. After this process, we first ran the quantization of input and output spaces, and then the space quantization as explained in Algorithm 16. As for offline learning, due to memory and time constraints, the algorithm did not reach a stopping criterion (neither machine determinism nor stability). This is because of the space complexity of the quantization mechanisms, which is overcome in the active (online) process by splitting the space in regions. Table 8 shows the normalized RMSE in prediction for both online and offline cases. The results obtained are close to the ones in simulation (Fig. 38), considering 200 or more training sequences. However, the SSMs we obtained are much more complex, taking into account the number of SSM states. In the real scenario, 100 ground truth sequences were used for testing.

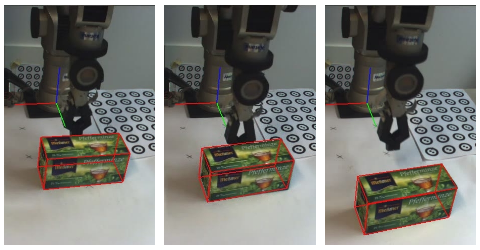
Considering the results and the learning parameters and conditions, we observe that apparently more sequences are needed to learn in a real scenario. Possible reasons are that the physical properties like friction parameters, object and arm properties, noisy features and other external aspects can have impact on the learnability of the environment, and they are poorly modeled by physics simulators. In previous work [30, 29], the scenario was set up to match the physical properties of a simulated scenario and to use the features from the physics simulator in the learning phase. Here, we use the visual features coming from the object tracker, which are always at least slightly noisy. However, this can be seen as an additional gain of our work, because we are trying to generalize object behaviors from noisy information which is a challenge in robotic learning scenarios. Additionally, we achieved this also by using incremental learning mechanisms.

Quantizers	Normalized RMS Error	Mean Confidence Interval	No. of States
Online	0.0116155743	±0.0004263449	12058
Offline	0.0120611487	±0.0004473294	12832

Table 8: Results for short-term prediction in a real scenario.

In Fig. 40 the prediction process during the pushing movement is shown. The results were obtained under different lighting conditions.


(a) Pushing to the right


(b) Pushing to the front

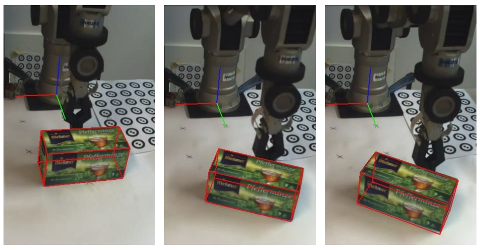

(c) Pushing to the left

Figure 40: Short-term prediction for pushing actions. Red countour depicts last prediction.

6.2.3.2 *Long-term prediction*

In Eq. 33 we saw the general form of the equation for predicting the distribution of a next state, given a previous state and an input symbol. We showed in Def. 7 that an input $\mathbf{i}(t) = \langle \mathbf{m}, \mathbf{a}(t), \mathbf{b}(t-1) \rangle$. Thus we have $x_k(t)$ whose corresponding latent variable k is obtained by

$$k(t) = \Lambda_i(\mathbf{i}(t)) = \Lambda_i(\langle \mathbf{m}, \mathbf{a}(t), \mathbf{b}(t-1) \rangle) \tag{37}$$

To obtain long-term predictions, we can use the inverse quantization function Λ_s^{-1} instead of using $\mathbf{b}(t-1)$. Thus, the index k can now be obtained by modifying the input as follows:

$$k(t) = \Lambda_i(\mathbf{i}(t)) = \Lambda_i(\langle \mathbf{m}, \mathbf{a}(t), \Lambda_s^{-1}(q(t-1)) \rangle) \tag{38}$$

For the first prediction, that is, to predict the second item in a sequence (i.e. for $t = 1$), we use the initial pose of the object which is $x_k(0)$ for some k, so that

$$k(1) = \Lambda_i(\mathbf{i}(1)) = \Lambda_i(\langle \mathbf{m}, \mathbf{a}(1), x_k(0) \rangle) \tag{39}$$

We then use Eqs. 33 and 35 with the modified $x_k(t)$ for prediction, starting with the initial perceived state $x_k(0)$.

Long-term predictions were also accurate although less precise than short-term predictions, as one might intuitively suspect. For comparison, in Table 9 we show the results for short and long-term prediction for 500 sequences in the simulated scenario. The SSMs learned by using 10 different sets of 500 sequences are used for both cases. In Table 10 we show long-term prediction results for online and offline experiments in the real scenario using 100 sequences, which can be compared with short-term prediction results presented in Table 8. The results show a higher error for long-term prediction and higher uncertainty, as mean confidence interval values show.

Quantizers	Normalized RMS Prediction Error	
	Short-term	Long-term
Online	0.0089 ± 0.00030	0.0135 ± 0.00038
Active	0.0122 ± 0.00040	0.0151 ± 0.00045

Table 9: Results for short and long-term prediction for 500 sequences in a simulated scenario.

Quantizers	Normalized RMS Error	Mean Confidence Interval
Online	0.02664327	±0.001506291
Offline	0.02728574	±0.002769571

Table 10: Results for long-term prediction in a real scenario.

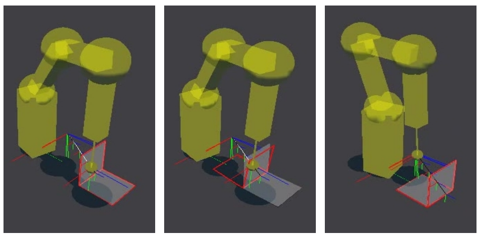
6.2.3.3 *Concept prediction*

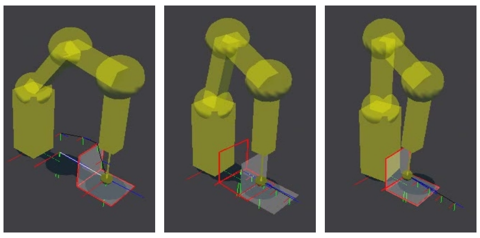
To test the ability of the CrySSMEx algorithm to infer probability distributions of more abstract patterns in the learning sequences, we applied the discretization explained in Chapter 3. As we mentioned there, 15 possible output values can be obtained. 5 of them correspond to one of the possible three abstract object motions (sliding, flipping over, tilting). These 5 values for each abstract motion correspond to the more fine-grained predicted object motion (going backwards, forwards, falling down, going up and not moving). We used 500 training sequences obtained during the trajectory prediction experiment and we ran CrySSMEx on them with the original input quantization already available (Section 6.2.2) and the new output quantization. We tested the prediction with 10 different sets of 500 sequences for both online and active density estimation cases. Table 11 shows misclassification statistics of object motions, where the confidence interval for the average misclassification percentage is calculated from the sample mean and variance, assuming a normal distribution with quantile $|z_{0.975}| = 1.96$. The results show that the prediction of fine-grained classifications is not satisfactory. However, although the fine-grained classifications were not accurately obtained, in most of the cases the output values obtained matched the corresponding coarse-grained classification. This leads us to the conclusion that at least an abstract pattern can be predicted by extracting SSMs from sensorimotor sequences.

Quantizers	Avg. Misclassification Percentage				No. of States
	Fine-grained	Mean Confidence Interval	Coarse-grained	Mean Confidence Interval	
Online	0.1815458	±0.0058833743	0.010212387	±0.0025574519	4174
Active	0.1986604	±0.004320878	0.02150257	±0.0044678776	4114

Table 11: Classification of abstract object motions.

The SSMs obtained for this categorization experiment can also be applied for trajectory prediction, though the predictions are approximate and coarse-grained because these SSMs possess less states. We illustrate these results in Figure 41.


(a) Predicting the flipping over affordance


(b) Predicting the sliding affordance

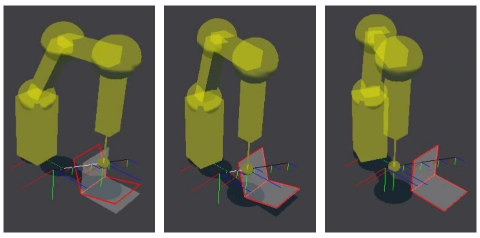

(c) Predicting the tilting affordance

Figure 41: Prediction of affordances by SSMs trained for abstract concept learning. Red contour depicts last prediction.

We also performed experiments for long-term classification prediction, by using the equations described in Section 6.2.3.2 with the proper output space discretization. The results are presented in Table 12 and again show that long-term predictions are still accurate.

Quantizers	Avg. Misclassification Percentage			
	Fine-grained	Mean Confidence Interval	Coarse-grained	Mean Confidence Interval
Online	0.1952707	±0.006386257	0.01561242	±0.003947912
Active	0.2106821	±0.005725652	0.02736386	±0.003201882

Table 12: Long-term prediction of object motion categories.

The results presented in this chapter show the ability of the SSMs to predict an object pose, given previous object and finger poses. Moreover, these predictions can be obtained in the long term. Thus, SSMs are able to predict object trajectories given a motor command. We show that abstract concepts can also be predicted accurately by using specialized output functions during SSM induction.

7 CONCLUSIONS

We investigated the problem of predicting the object behavior after a robot pushes it in terms of trajectory estimation and behavior categorization. Our approach uses Vector Quantization algorithms to discretize the sensorimotor space of action/object behavior pairs. After this, we infer probabilistic models which describe quantitative states and transitions which model cause-effect interactions between robotic arm actions and object movements.

More specifically, we applied an unsupervised learning algorithm (RobustGNG) to quantize the sensorimotor space after a new training sequence is available. This algorithm is incremental and allows to add or delete prototypes representing clusters in the probability distributions of these spaces in an online manner. Stopping criteria are crucial to decide the right number of latent variables. Information-theoretic criteria like Minimum Description Length were used for these purposes.

We also used a divide-and-conquer strategy to split the sensorimotor spaces into different regions, in order to accelerate convergence and deal with space complexity issues. Additionally, an active learning procedure for the quantization of sensorimotor spaces was implemented and some comparative results are presented, which show a potential for the applicability of active learning strategies for robotic learning tasks. Finally, the quantized spaces were used for obtaining qualitative models of object behavior and classification in the form of substochastic finite-state machines, by applying an improved version of the CrySSMEx algorithm. The prediction and classification ability of the learning algorithms were demonstrated here experimentally for both one-step ahead and long-term prediction cases.

The RobustGNG algorithm addresses some issues of previous implementations of GNG regarding online learning and efficient cluster identification. We managed to reduce the number of parameters compared to previous GNG implementations. Moreover, an online calculation of the error and a learning rate based on this quantity makes the algorithm suitable to be subsequently implemented as an online learning algorithm. For that purpose, the challenge is to implement strategies for identifying latent spaces (clusters) in an online manner. A possible solution is an approximate or online version of the minimum description length criterion, but other information-theoretic or probabilistic inference strategies might be applied.

In recent years considerable theoretical work has been carried out in online classification learning where active exploration can influence the training samples the learner is exposed to [37, 59]. Exploiting cluster structure in data has also been addressed in active learning models for classification tasks [14].

More theoretical or experimental work has to be done in order to compare offline, online and active learning approaches. For now, there is not enough evidence about the amount of instances needed after the active learning procedure becomes more efficient than simple random online selection of actions. Moreover, we applied active learning techniques for density estimation, which is a challenging task given the space complexity of the possible object behaviors produced. For this, CrySSMEx can also be helpful since we use conditional entropy for learning models and evaluating uncertainty. Active learning strategies can also be applied for the learning problem of identifying concepts or patterns from sequences.

For now, we use an offline algorithm (CrySSMEx) for inferring substochastic machines. We improved this algorithm to take into account information from the past to eliminate the ambiguities originated by the first-order Markov assumption. A next step in this work is to implement online strategies for building these probabilistic models in "real-time". After a sequence of object poses is obtained, information-theoretic approaches could be useful to decide which features are informative. A key aspect in this process pertains to the ability of an algorithm to correctly identify latent spaces in a constructive manner, after new information arrives.

In the future, the inferred probabilistic models obtained by employing CrySSMEx need to be tested to evaluate their suitability as planning tools. Planning is a final step in a developmental process which is relevant for robots when they learn and reproduce complex manipulation, spatial reasoning and language production tasks. Probabilistic finite-state machines are similar to hidden Markov models, where states are partially observable (see Section 2.5). HMMs in turn can be regarded as partially observed Markov decision processes (POMDPs) when actions (and eventually rewards) are present. There is a rich amount of information about decision-theoretic planning based on these Markov models [70].

An alternative approach is the use of SSMs as graphs with probabilities. Planning with graphs is possible through mechanisms such as evaluating reachability among different vertices of a graph. Additionally, it is useful to evaluate and compare different learning and inference approaches employed by various models such as HMMs, DBNs, DCRFs, among others (Section 2.5), which have commonalities and share similar applications. For instance, DCRFs possess advanced probabilistic inference methods, while SSMs make it possible to model incompleteness in their structure, but both deal with similar issues.

Other sequence learning models such as recurrent neural networks, like LSTMs, are useful as prediction tools, as we showed in Section 2.6. CrySSMEx is certainly a proper tool for extracting rules and qualitative states from them, as demonstrated in previous work. However, we showed that by using CrySSMEx and RobustGNG directly for learning and discretizing models of interactive dynamical systems, we obtain adequate results.

Active or online learning can also be accelerated by means of social interaction. As we already mentioned in Section 2.1, some works [68, 9] already address socially guided learning aspects. Autonomous learning can also benefit from imitation or demonstration learning environments [2]. Reciprocally, socially guided learning can benefit from active learning. In [10], the authors propose active learning applied to a socially guided learning environment, where the robot can query an external entity about areas of uncertainty in its hypothesis space. In

[56], the robot decides which features of the sensorimotor state vector are more salient to select actions or sensors only if their associated information gain is non-zero.

Autonomous robots should also be able to acquire abstract knowledge through verbal communication. There exist several stages of language or concept acquisition but there is still a lack of understanding about the interrelationships among different stages and processes. Thus, it is certainly useful to understand these developmental processes to implement them in robotic systems. As an example, there is an extensive literature on topics about verbal and motor lexicon acquisition through language games in robotic environments, including [61].

Additionally, robots need strategies for deciding the focus of attention in a visual scene, so that the space complexity is again reduced. One possible approach uses social cues from human tutors [68]. There are more technical approaches like segmenting a scene in order to identify possible places containing objects, as proposed in [7], or learning state representations to find a mapping from observations of the world to states that allow for choosing the right actions [27]. To make our systems more scalable, we call for the use of point clouds, optical flow estimation or similar strategies instead of model tracking systems. Traditionally, manipulation tasks have been addressed using algorithms specialized for certain problems. For instance, there are many approaches for grasping [32, 8, 46] including active learning and exploration, as we pointed out in Section 2.1. However, it is useful to develop more integrated systems which can generalize different behaviors or learning stages. To achieve that, a theory of developmental or cognitive robotics needs to be refined to strive for intelligent robots. For instance, the decisions on the motor actions to be performed have to be incorporated in a complex cognitive architecture which should be able to make decisions given different motivation drives and different contexts. A theoretical model of affordances or object-action complexes (see Section 2.2) needs to be incorporated in a cognitive architecture together with developmental approaches, including different aspects such as active/online learning, exploration, attention, planning and communication [74]. In summary, we argue that our approach can be incorporated in systems that involve social interaction, where developmental learning stages can make these systems more scalable by bringing more autonomy and self-motivation.

Last but not least, more features involving other object properties such as weight, shape or surface conditions have to be incorporated in the input space to make our systems scalable to real world tasks, as well as additional robotic control parameters such as velocity and tactile information.

BIBLIOGRAPHY

[1] A. Andreakis, N.V. Hoyningen-Huene, and M. Beetz. Incremental unsupervised time series analysis using merge growing neural gas. In *Proceedings of the 7th International Workshop on Advances in Self-Organizing Maps*, WSOM '09, pages 10–18, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-02396-5. doi: http://dx.doi.org/10.1007/978-3-642-02397-2_2. URL http://dx.doi.org/10.1007/978-3-642-02397-2_2.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009. ISSN 0921-8890. URL http://dx.doi.org/10.1016/j.robot.2008.10.024.

[3] I. Atil, N. Dag, S. Kalkan, and E. Sahin. Affordances and Emergence of Concepts. In *10th International Conference on Epigenetic Robotics*, 2010. URL http://kovan.ceng.metu.edu.tr/pub/pdf/atil-epirob-2010.pdf.

[4] H. Bischof, A. Leonardis, and A. Selb. MDL principle for robust vector quantisation. *Pattern Analysis Applications*, 2(1):59–72, 1999. URL http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s100440050015.

[5] C.M. Bishop. GTM through time. *Fifth International Conference on Artificial Neural Networks*, 1(2):111–116, 1997. ISSN 19393210. doi: 10.1049/cp:19970711. URL http://link.aip.org/link/IEECPS/v1997/iCP440/p111/s1&Agg=doiresearch.microsoft.com/../Bishop-GTMTT-IEE-97.pdf.

[6] C.M. Bishop. *Pattern recognition and machine learning*. Information science and statistics. Springer, New York

[u.a.], 5. (corr. print.) edition, 2007. ISBN 0-387-31073-8 ; 978-0-387-31073-2. URL http://swbplus.bsz-bw.de/bsz28047668xinh.htm.

[7] J. Bohg, M. Johnson-Roberson, M. Björkman, and D. Kragic. Strategies for multi-modal scene exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[8] M. Buckmann, R. Gaschler, S. Höfer, D. Loeben, P. A. Frensch, and O. Brock. Learning to explore the structure of kinematic objects in a virtual environment. *Frontiers in Psychology*, 6(374), April 2015.

[9] M. Cakmak, N. DePalma, A. L. Thomaz, and R. I. Arriaga. Effects of social exploration mechanisms on robot learning. In *18th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2009, Toyama International Conference Center, Japan, September 27 - October 2, 2009*, pages 128–134, 2009. URL http://dx.doi.org/10.1109/ROMAN.2009.5326168.

[10] C. Chao, M. Cakmak, and A. L. Thomaz. Transparent active learning for robots. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 317–324, March 2010.

[11] C. Chatfield. *Time-series forecasting*. Chapman \& Hall/CRC, 2000. ISBN 1584880635. URL http://books.google.com/books?id=fLlGsTFb21EC&pgis=1.

[12] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[13] Ö. Şimşek and A.G. Barto. An intrinsic reward mechanism for efficient exploration. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 833–840, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: http://doi.acm.org/10.1145/1143844.1143949.

[14] S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. *Proceedings of the 25th International Conference on Machine Learning (2008)*, 307(NASA SP-192):208–215, 2008. URL http://portal.acm.org/citation.cfm?doid=1390156.1390183.

[15] P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, 2005. ISSN 00313203. doi: 10.1016/j.patcog.2004.03.020. URL http://linkinghub.elsevier.com/retrieve/pii/S0031320305000233.

[16] B. Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, pages 625–632. MIT Press, 1995.

[17] J.J. Gibson. The theory of affordances. In R. Shaw and J. Bransford, editors, *Perceiving, Acting, and Knowing: Toward an Ecological Psychology*, pages 67–82. Lawrence Erlbaum, 1977.

[18] H. Grabner. *On-line Boosting and Vision*. PhD thesis, Graz University of Technology, 2008.

[19] S. Graf and H. Luschgy. *Foundations of quantization for probability distributions*. Lecture notes in mathematics ; 1730. Springer, Berlin, 2000. ISBN 3-540-67394-6. URL http://swbplus.bsz-bw.de/bsz085161799cov.htm; http://www.gbv.de/dms/ilmenau/toc/312760310.PDF.

[20] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. PhD thesis, Technische Universität München, July 2008. URL http://www6.in.tum.de/pub/Main/Publications/Graves2008c.pdf.

[21] R.M. Gray and R.A. Olshen. Vector quantization and density estimation. In *In SEQUENCES97*, page http://wwwisl. stanf, 1997.

[22] P. Grünwald. *The Minimum Description Length Principle*, volume 5. MIT Press, 2007. URL http://eprints.pascal-network.org/archive/00003327/.

[23] F.H. Hamker. Life-long learning cell structures–continuously learning without catastrophic interference. *Neural Networks*, 14:551–573, 2001. doi: 10.1016/S0893-6080(01)00018-1.

[24] T. Hermans, J. M. Rehg, and A. F. Bobick. Decoupling Behavior, Perception, and Control for Autonomous Learning of Affordances. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2013.

[25] H. Jacobsson. The crystallizing substochastic sequential machine extractor - CrySSMEx. *Neural Computation*, 18(9): 2211–2255, 2006.

[26] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999. ISSN 03600300. doi: 10.1145/331499.331504. URL http://portal.acm.org/citation.cfm?doid=331499.331504.

[27] R. Jonschkowski and O. Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39(3):407–428, 2015. ISSN 0929-5593.

[28] F. Kaplan and P.-Y. Oudeyer. Motivational principles for visual know-how development. In C.G. Prince, L. Berthouze, H. Kozima, D. Bullock, G. Stojanov, and C. Balkenius, editors, *Proceedings of the 3rd Epigenetic Robotics workshop : Modeling cognitive development in robotic systems*, volume 101 of *Lund University Cognitive Studies*, pages 72–80, 2003.

[29] M. Kopicki. *Prediction learning in robotic manipulation*. PhD thesis, University of Birmingham, April 2010.

[30] M. Kopicki, J. Wyatt, and R. Stolkin. Prediction learning in robotic pushing manipulation. In *Proceedings of the 14th IEEE International Conference on Advanced Robotics (ICAR 2009)*, Munich, Germany, June

2009. URL http://cogx.eu/data/cogx/publications/KopickiWyattStolkinICAR2009.pdf.

[31] M. Kopicki, S. Zurek, R. Stolkin, T. Mörwald, and J. Wyatt. Learning to predict how rigid objects behave under simple manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA11)*, May 2011. URL http://www.cs.bham.ac.uk/~msk/pub/icra2011.pdf.

[32] D. Kraft, R. Detry, N. Pugeault, E. Baseski, F. Guerin, J. Piater, and N. Krueger. Development of Object and Grasping Knowledge by Robot Exploration. *IEEE Transactions on Autonomous Mental Development*, 2(99):1, 2010. URL http://eprints.pascal-network.org/archive/00007163/.

[33] S.C. Kremer. Spatiotemporal connectionist networks: A taxonomy and review. *Neural Computation*, 13(2):249–306, 2001. URL http://www.mitpressjournals.org/doi/abs/10.1162/089976601300014538.

[34] M. Kristan and A. Leonardis. Online discriminative kernel density estimation. In *Pattern Recognition, International Conference on*, volume 0, pages 581–584, Los Alamitos, CA, USA, 2010. IEEE Computer Society. doi: http://doi.ieeecomputersociety.org/10.1109/ICPR.2010.147.

[35] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010. URL http://eprints.pascal-network.org/archive/00008026/.

[36] N. Krueger, J. Piater, F. Wörgötter, C.W. Geib, R.P.A. Petrick, M. Steedman, A. Ude, T. Asfour, D. Kraft, D. Omrcen, B. Hommel, A. Agostini, D. Kragic, J.-O. Eklundh, V. Krueger, C. Torras, and R. Dillmann. A Formal Definition of Object-Action Complexes and Examples at Different Levels of the Processing Hierarchy. *Archives of clinical neuropsychology the official journal of the National Academy of Neuropsychologists*, 21(1):1–39, 2009. ISSN 08876177. doi:

10.1016/j.acn.2005.07.006. URL http://wwwiaim.ira.uka.de/pacoplus/download/OAC-Definition.pdf.

[37] L. Li, M. L. Littman, and T. J. Walsh. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 568–575, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390228. URL http://doi.acm.org/10.1145/1390156.1390228.

[38] T. Martinetz. Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps. In Stan Gielen and Bert Kappen, editors, *Proc. ICANN'93, Int. Conf. on Artificial Neural Networks*, pages 427–434, London, UK, 1993. Springer.

[39] T. Martinetz and K. Schulten. A "Neural-Gas" Network Learns Topologies. *Artificial Neural Networks*, I:397–402, 1991.

[40] T.M. Martinetz, S.G. Berkovich, and K.J. Schulten. 'neural-gas' network for vector quantization and its application to time-series prediction. *Neural Networks, IEEE Transactions on*, 4(4):558 –569, jul 1993. ISSN 1045-9227. doi: 10.1109/72.238311.

[41] T. Mörwald, M. Zillich, J. Prankl, and M. Vincze. Self-monitoring to improve robustness of 3d object tracking for robotics. In *IEEE International Conference on Robotics and Biomimetics*, Phuket, Thailand, Dec 2011. doi: 10.1109/ROBIO.2011.6181734. URL http://users.acin.tuwien.ac.at/tmoerwald/files/moerwald2011self.pdf.

[42] J. Mugan and B. Kuipers. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development (TAMD)*, 4 (1):70–86, 2012.

[43] K.P. Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, 2002.

[44] D.A. Norman. *The Design of Everyday Things*. Number no. 842 in The Design of Everyday Things. Bantam Doubleday Dell Publishing Group, 1988. ISBN 9780385267748. URL http://books.google.de/books?id=b09jQgAACAAJ.

[45] I. Olier and A. Vellido. Advances in clustering and visualization of time series using GTM through time. *Neural Networks*, 21(7):904–913, 2008. URL http://www.ncbi.nlm.nih.gov/pubmed/18653311.

[46] D. Omrčen, C. Böge, T. Asfour, A. Ude, and R. Dillmann. Autonomous acquisition of pushing actions to support object grasping with a humanoid robot. In *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, pages 277–283, Dec 2009.

[47] P.-Y. Oudeyer, F. Kaplan, and V.V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[48] R. Petrick, D. Kraft, N. Krüger, and M. Steedman. Combining cognitive vision, knowledge-level planning with sensing, and execution monitoring for effective robot control. In *ICAPS 2009 Workshop on Planning and Plan Execution for RealWorld Systems*, pages 58–65, 2009. URL http://homepages.inf.ed.ac.uk/rpetrick/papers/icaps2009ws-planexec-slides.pdf.

[49] A.K. Qin and P.N. Suganthan. Robust growing neural gas algorithm with application in cluster analysis. *Neural Networks*, 17(8-9):1135 – 1148, 2004. ISSN 0893-6080. doi: DOI:10.1016/j.neunet.2004.06.013. URL http://www.sciencedirect.com/science/article/pii/S0893608004001662. New Developments in Self-Organizing Systems.

[50] K. Qin and N. Suganthan. Enhanced neural gas network for prototype-based clustering. *Pattern Recognition*, 38(8):1275–1288, 2005. ISSN 00313203. doi: 10.1016/

j.patcog.2004.12.007. URL http://linkinghub.elsevier.com/retrieve/pii/S0031320305000208.

[51] B. Ridge, D. Skočaj, and A. Leonardis. Unsupervised learning of basic object affordances from object properties. In *Proceedings of the Fourteenth Computer Vision Winter Workshop (CVWW)*, pages 21–28, Eibiswald, Austria, February, 4–6 2009. URL http://cogx.eu/data/cogx/publications/ridgeCVWW09.pdf.

[52] B. Ridge, D. Skocaj, and A. Leonardis. Self-supervised cross-modal online learning of basic object affordances for developmental robotic systems. In *Robotics and Automation ICRA 2010 IEEE International Conference on*, pages 5047–5054. IEEE, 2010. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5509544.

[53] S. Roa and G.-J. Kruijff. Robust Vector Quantization for Inference of Substochastic Sequential Machines. Technical report, DFKI GmbH, 2011.

[54] S. Roa and G.-J.M. Kruijff. Offline and active gradient-based learning strategies in a pushing scenario. In *19th European Conference on Artificial Intelligence 2010: 3rd International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2010)*, 2010.

[55] E. Sahin, M. Cakmak, M.R. Dogar, E. Ugur, and G. Ucoluk. To afford or not to afford: A new formalization of affordances toward affordance-based robot control. *Adaptive Behavior*, 15(4):447–472, 2007. URL http://adb.sagepub.com/cgi/doi/10.1177/1059712307084689.

[56] J. Saunders, C. L. Nehaniv, K. Dautenhahn, and A. Alissandrakis. Self-imitation and environmental scaffolding for robot teaching. *International Journal of Advanced Robotics Systems*, 4(1):109–124, 2007.

[57] J. Schmidhuber. Curious model-building control systems. In *In Proc. International Joint Conference on Neural Networks, Singapore*, pages 1458–1463. IEEE, 1991.

[58] J. Scholz and M. Stilman. Combining motion planning and optimization for flexible robot manipulation. In *Humanoids*, pages 80–85. IEEE, 2010. ISBN 978-1-4244-8688-5. URL http://dblp.uni-trier.de/db/conf/humanoids/humanoids2010.html#ScholzS10.

[59] D. Skočaj, M. Majnik, M. Kristan, and A. Leonardis. Comparing different learning approaches in categorical knowledge acquisition. In *Proceedings of the 2012 Computer Vision Winter Workshop (CVWW)*, Feb 2012. URL http://cogx.eu/data/cogx/publications/skocajCVWW12.pdf.

[60] A. Sloman. Polyflaps as a domain for perceiving, acting and learning in a 3-D world. In *Position Papers for 2006 AAAI Fellows Symposium*, Menlo Park, CA, 2006. AAAI. URL http://www.cognitivesystems.org/publications/Fellows16.pdf.

[61] L. Steels and M. Hild, editors. *Language Grounding in Robots*. Springer, New York, 2012. ISBN 978-1-4614-3063-6. URL http://www.springer.com/computer/ai/book/978-1-4614-3063-6.

[62] A. Stoytchev. Learning the affordances of tools using a behavior-grounded approach. In *Proceedings of the 2006 International Seminar: Towards affordance-based robot control*, pages 140–158, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3-540-77914-0, 978-3-540-77914-8. URL http://dl.acm.org/citation.cfm?id=1787357.1787367.

[63] A. Stoytchev. Some basic principles of developmental robotics. *IEEE Trans. on Auton. Ment. Dev.*, 1(2):122–130, August 2009. ISSN 1943-0604. doi: 10.1109/TAMD.2009.2029989. URL http://dx.doi.org/10.1109/TAMD.2009.2029989.

[64] M. Strickert, B. Hammer, and S. Blohm. Unsupervised recursive sequence processing. *Neurocomputing*, 63 (April):69–97, 2005. ISSN 09252312. doi: 10.1016/j.neucom.2004.01.190. URL http://linkinghub.elsevier.com/retrieve/pii/S0925231204003170.

[65] C. Sutton, K. Rohanimanesh, and A. McCallum. Dynamic Conditional Random Fields : Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. *Journal of Machine Learning Research*, 8(2):693–723, 2004. ISSN 15324435. doi: 10.1145/1015330.1015422. URL http://portal.acm.org/citation.cfm?id=1015422.

[66] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[67] R.S. Sutton, D. Precup, and S.P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999. URL citeseer.ist.psu.edu/sutton99between.html.

[68] A.L. Thomaz. *Socially Guided Machine Learning*. PhD thesis, Massachusetts Institute of Technology, May 2006.

[69] S. Thrun. *Handbook of Brain Science and Neural Networks*, chapter Exploration in active learning. MIT Press, 1995.

[70] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[71] P. Tino and M. Köteles. Extracting finite-state representations from recurrent neural networks trained on chaotic symbolic sequences. *IEEE Transactions on Neural Networks*, 10(2):284–302, 1999. URL http://dx.doi.org/10.1109/72.750555http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.16.3772&rep=rep1&type=pdf.

[72] E. Ugur, E. Sahin, and E. Oztop. Affordance learning from range data for multi-step planning. In *Ninth International Conference on Epigenetic Robotics (EpiRob09)*, 2009. URL http://www.kovan.ceng.metu.edu.tr/~{}erol/publications/pdf/Ugur-Epirob-2009.pdf.

[73] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[74] J.L. Wyatt, A. Aydemir, M. Brenner, M. Hanheide, N. Hawes, P. Jensfelt, M. Kristan, G.-J.M. Kruijff, P. Lison, A. Pronobis, K. Sjöö, A. Vrečko, H. Zender, M. Zillich, and D. Skočaj. Self-understanding and self-extension:a systems and representational approach. *IEEE Transactions on Autonomous Mental Development (TAMD), Special Issue on Representations and Architectures for Cognitive Systems*, 2(4):282–303, December 2010. doi: 10.1109/TAMD.2010.2090149. URL `http://www.pronobis.pro/publications/wyatt2010tamd`.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

EIDESSTATTLICHE VERSICHERUNG

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In-noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, Februar 2016

Sergio Roa Ovalle, M.Sc.