Technische Universität München
Fakultät für Elektrotechnik und Informationstechnik
Lehrstuhl für Kognitive Systeme

A New Developmental Cognitive Architecture for the Autonomous Acquisition of Sensory-Motor Skills on Humanoid Robots

Dipl.-Ing. Univ. Erhard Wieser

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik
der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing. Klaus Diepold

Prüfer der Dissertation:

1. Prof. Gordon Cheng, Ph.D.

2. Prof. Jun Tani, Ph.D.

Die Dissertation wurde am 12.03.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 04.11.2019 angenommen.

To my family

Abstract

The sensory-motor skills that humans progressively acquire even during their first months and later throughout their life are impressive and far beyond the capabilities of today's robots. In particular, hand-eye coordination is the basis for many higher-level skills. While humans easily acquire coordination ability on their own at an early stage, robots still seem to have difficulties with it. Pre-programming the coordination, for example through a given kinematics, leads to specific and precise execution of skill but suffers from the lack of *adaptivity and robustness to sensory-motor deficits*. The alternative is an autonomous learning of coordination skills inspired by recent neurobiological findings.

This thesis deals with the problem of *autonomous acquisition of sensory-motor coordination* on humanoid robots. The problem is related to the *poverty of stimulus* that constrains developmental agents, biological and artificial, when they have to generalize from a minimum amount of data.

This thesis approaches the problem by extending a *predictive coding* method and by integrating it into a framework of *biologically-inspired algorithms*. The resulting system is a new developmental cognitive architecture that enables a robot to progressively learn increased levels of sensory-motor coordination.

Besides a review of the limitations of state-of-the-art methods, the contributions encompass three major parts:

The first part deals with the extension of the multiple timescale recurrent neural network. The original version requires additional networks to pre- and postprocess input-output data. It also requires a careful setting of hyperparameters, such as timescales and number of context neurons, for a successful learning. So far, this parameterization has been done manually by a human expert. The proposed *evolutionary optimized multiple timescale recurrent neural network* has an uniform neural activation and is combined with an analytical pre- and postprocessing schema that makes additional networks dispensable. The training method also contains an early stopping schema to avoid overfitting. The key feature of this proposed network is its ability to *autonomously estimate its hyperparameters*. The benefit is a reduction of parameterization effort when applying the network to learn a given set of teaching data.

Building on the proposed network, the second part deals with the proposed *predictive action selector* that integrates action generation and action selection into *one* framework, yielding a greater flexibility in movement generation compared to state-of-the-art action selection like in the iCub cognitive architecture. The predictive action selector goes beyond the limit of state-of-the-art predictive coding methods that firstly always require a teacher providing data, and secondly cannot coordinate the robot's motion depending on moving targets. The predictive action selector *autonomously bootstraps* sensory-motor coordination from a minimum amount of data that are self-generated through a constrained degree of freedom exploration. Moreover, the predictive action selector can *improve* in coordination by reducing the skill execution time, for example yielding a faster reaching.

The third part deals with the connection of the predictive action selector to a multi-layered perceptron and symbol-based algorithms in order to extend the prediction capability to a longer timespan. The resulting system is the proposed *self-verifying cognitive architecture* that operates with *loops of imaginary trial and physical trial* of actions. The purpose of these loops is the self-verification of sensory samples and the action outcome. This type of verification yields an increase of robustness in skill execution, such as blindfolded reaching, and it yields the ability to adapt to different robot platforms.

The proposed methods are validated on two robots, NAO and TOMM, that differ in their size, sensors, and motors. Experimental results confirm the *robustness to visual occlusion* and the adaptation to different robots by *scaling to the number of degree of freedom* as well as by *adapting to motor backlash*.

The results imply that a progressive improvement of spatiotemporal prediction, combined with self-verification, yields adaptability and robustness in the sensory-motor domain and is thereby beneficial to future developmental agents.

Kurzfassung

Die sensomotorischen Fähigkeiten, die Menschen sich fortlaufend sogar in ihren ersten Monaten und später ihr ganzes Leben hindurch aneignen, sind beeindruckend und weit über den Fähigkeiten heutiger Roboter. Hand-Augen Koordination ist vor allem die Grundlage für viele höhere Fähigkeiten. Während Menschen sich die Koordinationsfähigkeit leicht in einem frühen Stadium selbstständig aneignen, scheinen Roboter noch Schwierigkeiten damit zu haben. Eine Vorprogrammierung der Koordination, zum Beispiel durch eine gegebene Kinematik, führt zu spezifischer und präziser Ausführung einer Fertigkeit, leidet aber an einem Mangel an *Adaptivität und Robustheit gegenüber sensomotorischen Defiziten*. Die Alternative ist ein autonomes Lernen von Koordinationsfähigkeiten, inspiriert durch jüngste neurobiologische Erkenntnisse.

Diese Doktorarbeit behandelt das Problem der *autonomen Aneignung von sensomotorischer Koordination* auf humanoiden Robotern. Das Problem ist verbunden mit der *Armut an Stimulus*, die entwicklungsfähige biologische und künstliche Agenten beschränkt, wenn sie von einer minimalen Menge an Daten verallgemeinern müssen.

Diese Doktorarbeit geht das Problem an, indem sie eine Methode der *prädiktiven Codierung* erweitert und diese in eine Rahmenstruktur von *biologisch inspirierten Algorithmen* integriert. Das resultierende System ist eine neue sich entwickelnde kognitive Architektur, die es einem Roboter ermöglicht, fortlaufend ein höheres Niveau an sensomotorischer Koordination zu erlernen.

Neben einem Rückblick auf die Grenzen der heutigen technischen Methoden, umfassen die Beiträge drei Hauptteile:

Der erste Teil behandelt die Erweiterung des multiplen Zeitskalen rekurrenten neuronalen Netzes. Die Originalversion benötigt zusätzliche Netze, um Eingangs- und Ausgangsdaten vor- und nach zu verarbeiten. Sie benötigt auch eine sorgfältige Einstellung der Hyperparameter, wie zum Beispiel Zeitskalen und Anzahl der Kontextneuronen, für ein erfolgreiches Lernen. Bis jetzt wurde diese Parametrisierung manuell durch einen menschlichen Experten vorgenommen. Das vorgeschlagene *evolutionär optimierte multiple Zeitskalen rekurrente neuronales Netz* hat eine einheitliche neuronale Aktivierung und ist mit einem analytischen Vor- und Nachverarbeitungsschema kombiniert, welches zusätzliche Netze verzichtbar macht. Die Trainingsmethode beinhaltet auch ein Frühstoppschema, um eine Überanpassung zu vermeiden. Das Hauptmerkmal dieses vorgeschlagenen Netzes ist seine Fähigkeit, seine *Hyperparameter autonom zu schätzen*. Der Vorteil ist eine Reduzierung des Aufwandes der Parametrisierung, falls das Netz angewendet wird, um einen gegebenen Datensatz zu lernen.

Auf dem vorgeschlagenen Netz aufbauend, behandelt der zweite Teil den vorgeschlagenen *prädiktiven Aktionsselektor*, welcher Aktionsgenerierung und Aktionsselektion in *eine* Rahmenstruktur integriert, die eine größere Flexibilität in der Bewegungsgenerierung ergibt, verglichen mit der heutigen Aktionsselektion wie in der iCub kognitiven Architektur. Der prädiktive Aktionsselektor geht über die Grenzen der heutigen prädiktiven Codierungsmethoden, die erstens immer einen Lehrer für die Bereitstellung der Daten brauchen, und zweitens die Bewegung des Roboters nicht in Abhängigkeit von sich bewegenden Zielen koordinieren können. Der prädiktive Aktionsselektor *generiert* die sensomotorische Koordination *autonom* aus einer minimalen Menge an Daten, die selbst erzeugt werden durch eine eingeschränkte Exploration von Freiheitsgraden. Außerdem kann der prädiktive Aktionsselektor sich in der Koordination *verbessern*, indem er die Ausführungszeit einer Fertigkeit reduziert, was zum Beispiel ein schnelleres Greifen ergibt.

Der dritte Teil behandelt die Verbindung des prädiktiven Aktionsselektors mit einem Mehrschicht-Perzeptron und Symbol-basierten Algorithmen, um die Prädiktionsfähigkeit auf eine längere Zeitspanne zu erweitern. Das resultierende System ist die vorgeschlagene *selbstverifizierende kognitive Architektur*, die mit *Schleifen von imaginärem Versuch und physischem Versuch* von Aktionen operiert. Der Zweck dieser Schleifen ist die Eigenprüfung von sensorischen Proben und vom Aktionsausgang. Diese Art der Überprüfung ergibt eine Zunahme der Robustheit bei der Ausführung von Fertigkeiten, wie zum Beispiel blindes Greifen, und ergibt die Fähigkeit, sich an verschiedene Roboterplattformen anzupassen.

Die vorgeschlagenen Methoden sind auf zwei Robotern validiert, NAO und TOMM, die sich in ihrer Größe, Sensoren, und Motoren unterscheiden. Experimentelle Resultate bestätigen die *Robustheit gegenüber visueller Verdeckung* und die Anpassung an verschiedene Roboter durch die *Skalierung auf die Anzahl der Freiheitsgrade*, sowohl als auch durch die *Anpassung an das Getriebespiel*.

Die Resultate implizieren, dass eine kontinuierliche Verbesserung der raumzeitlichen Prädiktion, kombiniert mit Eigenprüfung, eine Anpassungsfähigkeit und Robustheit im sensomotorischen Bereich ergibt und damit vorteilhaft für künftige entwicklungsfähige Agenten ist.

Acknowledgements

First and foremost, I thank Prof. Dr. Gordon Cheng for the opportunity to conduct my Ph.D. research at his institute. I am grateful for his continuous supervision and support. He gave me enough time to find and follow my own research interests.
I also thank the entire ICS team for their support throughout the years.
My thanks go to Dr. Pablo Lanillos Pradas, Dr. Karinne Ramírez Amaro, Dr. Emmanuel Carlos Dean-Leon, Dr. Mohsen Kaboli, and Stefan Ehrlich for the scientific conversations. I thank Quentin Leboutet and Ilya Dianov for our joint efforts that we have put into teaching. Thanks go to Constantin Uhde, Julio Rogelio Guadarrama-Olvera, Florian Bergner, Jasmin Isabella Kajopoulos, Zied Tayeb, Katharina Stadler, Sebastian Stenner, and Wolfgang Burger, as well as to my former colleagues Dr. Ewald Lutscher, Dr. Andreas Holzbach, and Dr. John Nassour.
I thank Ilona Nar-Witte, Wibke Borngesser, and Brigitte Rosenlehner for their support in administrative formalities.
Finally, special thanks go to my family and all my friends.

“In general, we’re least aware of what our minds do best.”

— Marvin Minsky (*The Society of Mind*, 1986, p. 29)

Contents

List of Publications

Parts of this dissertation have been published in the following peer-reviewed journals and conference proceedings including workshop:

Journal papers

- **Erhard Wieser** and Gordon Cheng. EO-MTRNN: Evolutionary optimization of hyperparameters for a neuro-inspired computational model of spatiotemporal learning. *Biological Cybernetics*, 2020. `https://doi.org/10.1007/s00422-020-00828-8`.

- **Erhard Wieser** and Gordon Cheng. A self-verifying cognitive architecture for robust bootstrapping of sensory-motor skills via multipurpose predictors. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):1081–1095, 2018.

Conference papers

- Wolfgang Burger*, **Erhard Wieser***, Emmanuel Dean-Leon, and Gordon Cheng. A scalable method for multi-stage developmental learning for reaching. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Lisbon, Portugal, pages 60–65, 2017.
  *Wolfgang Burger and Erhard Wieser had an equal contribution to the paper.

- **Erhard Wieser** and Gordon Cheng. Progressive learning of sensory-motor maps through spatiotemporal predictors. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Cergy-Pontoise, Paris, France, pages 43–48, 2016.

- **Erhard Wieser** and Gordon Cheng. Predictive action selector for generating meaningful robot behaviour from minimum amount of samples. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Genoa, Italy, pages 139–145, 2014.

Workshop paper

- **Erhard Wieser** and Gordon Cheng. Forming goal-directed memory for cognitive development. In *Proceedings of Humanoids 2012 Workshop on Developmental Robotics: Can developmental robotics yield human-like cognitive abilities?*, pages 38–39. Workshop at the *IEEE International Conference on Humanoid Robots*, Osaka, Japan, 2012.

List of Figures

List of Tables

List of Algorithms

List of Acronyms

Every listed acronym is expanded on its first use, indicated by the page number.

AHE	autonomous hyperparameter estimation	6
AI	artificial intelligence	4
BPTT	backpropagation through time	25
CPU	central processing unit	80
CTRNN	continuous time recurrent neural network	36
DE	differential evolution	52
DOF	degree of freedom	3
EO-MTRNN	evolutionary optimized multiple timescale recurrent neural network	6
FA-DE	fuzzy adaptive differential evolution	52
FOV	field of view	3
FP	feature predictor	91
GPU	graphics processing unit	80
HHOP	higher-order Hopfield network	8
LITAPT	loops of imaginary trial and physical trial	8
LSTM	long short-term memory	48
M1	primary motor	116
MLP	multi-layered perceptron	9
MSE	mean squared error	58
MST	medial superior temporal	116

MTRNN	multiple timescale recurrent neural network	6
PAS	predictive action selector	7
PB	parametric bias	24
P control	proportional control	102
PMd	dorsal premotor	116
PSO	particle swarm optimization	52
RMSD	root-mean-square deviation	120
RNN	recurrent neural network	24
ROS	robot operating system	94
SA-DE	self-adaptive differential evolution	52
SMP	self-motion predictor	91
SOM	self-organizing map	20
SVCA	self-verifying cognitive architecture	6
VFCs	visual feature cells	95

1. Chapter

Introduction

Humans progressively acquire impressive capabilities even during the first months and years of post-natal development, and then throughout their life. While still in infancy, they learn to perform countless skills with different levels of complexity such as opening a drawer, playing yo-yo, and painting a picture. Despite the fact that the acquired complex skills vary from human to human, a common ground is the set of capabilities learned during the first months of life, namely the *sensory-motor coordination* skills. A human cannot acquire a complex skill without the presence of an earlier simpler coordination skill. The same is true for developmental robots [217]. Thus, it is worth to investigate the early development of sensory-motor capabilities that are the foundation of subsequent higher-level capabilities later on.

While the variety of the sensory-motor abilities is tremendous, it follows a developmental structure, in which complex abilities are learned based on the availability of less complex ones that have been acquired at earlier stages. For instance, saccadic eye movements are already present in a very early stage, reaching ability is learned in the first months while it is continuously refined, leading to the emergence of hand-eye coordination that is robust to disturbances, e.g. occlusions of the hand or target objects [211]. During these months, the brain continuously learns a model of the surrounding environment, including the behaviour of external objects and their relation to each other. This knowledge is beneficial for improved sensory-motor coordination that involves the prediction and coordination of one's own movement (ego-motion) as well as the prediction of the motion of external entities.

An illustrative example is an infant building a tower out of toy bricks. This is a relative complex skill that depends on more simple ones like reaching for objects. The ability of prediction facilitates the skill execution and influences the action selection. The infant would not place a brick on top of another one, if it predicts that the resulting configuration is not stable and would fall over. Prediction also makes the actions robust to environmental disturbances like visual occlusion, e.g. when the target brick is getting covered or occluded by another object during the reaching phase.

The reproduction of prediction capability on robots, along with aspects of skill development, would make the robots more adaptive and robust to environmental disturbances.

In Section 1.1, I continue to elaborate the aforementioned thoughts and provide a motivation for the research presented in this thesis. I describe the problem and deduce a set of research questions in Section 1.2. In order to tackle the problem and to answer the questions, a new approach is required that is presented in Section 1.3. Implementing and evaluating the new approach lead to several scientific contributions that I describe in Section 1.4. The thesis outline is presented in Section 1.5.

Finally, Section 1.6 lists own publications that contain parts of the material presented in this thesis.

1.1. Motivation

In this thesis, the research is motivated by humans' autonomous acquisition of sensory-motor coordination skills through progressive learning. These first sensory-motor capabilities are the basis for the subsequent learning or acquisition of higher-level capabilities that are acquired during the first years of life, including cognitive skills [108], [172], [138]. In particular, the emergence of hand-eye coordination plays a key role, along with the cognitive abilities related to hand-eye coordination such as predicting the location of own body parts and predicting the location of external objects.

Infants acquire these early coordination skills autonomously through interaction with their environment. In infancy, the brain is highly adaptive [78] and its plasticity is influenced by environmental and social interaction. This interaction shapes the infant's brain and forms its developmental pathway [172], [198].

Inspired by humans' development of coordination skills, a robot should develop its sensory-motor coordination on its own through interactive processes. Starting from zero-knowledge about coordination, the robot should learn how to coordinate its body parts in a meaningful way and how to further extend and improve its coordination. In other words, the robot should learn skills along a developmental pathway. The benefit of this developmental approach to learning is its adaptation capability, especially adaptation to the robot's morphology. One would expect that a developmental cognitive architecture can learn to coordinate different types of robot morphologies. For example, it can learn to control a robot with arms having precise motors (e.g. industrial robot arms) as well as a robot with imprecise motors.

In this context, sensory-motor learning yields an important cognitive ability: spatiotemporal prediction. Since prediction is a key operating principle of the human cortex [59], [60], the robot's cognitive architecture has to feature predictive capability, which should be present in many different scenarios of the robot's learning phases. A benefit of the prediction capability is that it yields a robust interaction. For example, incomplete sensory information can be compensated for or trajectories of external objects for interaction can be computed in advance.

One can imagine a future scenario, in which a robot is equipped with a cognitive architecture that does not yet support a wide variety of capabilities at the beginning of its operation but has all the necessary mechanisms that enable it to learn skills autonomously as well as through a teacher. While the robot would learn fundamental sensory-motor skills on its own, it would later acquire complex procedural skills through imitating the actions shown by a human teacher.

The scope of this thesis is set by the former case, i.e. the *autonomous learning of coordination skills* that is facilitated through *prediction capability*. But what are the benefits of this kind of autonomous learning and prediction? The learning process generates the ability to predict, which in turn leads to *adaptive* and *robust* execution of a skill. By progressively learning increased levels of sensory-motor coordination, the robot could achieve the needed adaptivity for changes in its environment, such as visual occlusion.

In order to substantiate this argumentation, I provide the following example scenarios that illustrate the desired characteristics of a new cognitive architecture:

1. **Robustness to loss of sensory data:** The robot is executing a reaching action as part of a pick and place task. Suddenly, its cameras are covered or its cameras are failing entirely and the robot has lost its vision for the entire interaction lasting for hours, although the location of each target object for the picking has not changed. In this situation, the robot would have to be able to perform the skill blindfolded. For such robust performance, the robot would have to predict its end-effector position in the visual space. The same capability is required when the cameras are operational, but the end-effector gets covered or occluded for a long period of time.

2. **Facilitation of robust interaction:** The robot is confronted with partial occlusion or covering of a moving target object in its field of view (FOV). This can happen in two types of interaction: interaction with people and interaction with the environment. For the interaction with people, an example is a robot and an infant playing a game that includes partial occlusion of the target object (e.g. a toy). For the interaction with the environment, an example is a robot that is operating at an assembly line where the moving parts get occluded temporarily. In each of these cases, the target object that the robot is focusing on may become occluded for some seconds due to covering by another object. Instead of searching the visual space or stopping the execution of motion, the robot would have to be able to predict where the target object would reappear in the next moment in order to produce continuous, smoother tracking of the target, yielding a robust interaction.

3. **Scalability to the number of degree of freedom:** The robot is exposed to changes in the number of degree of freedom (DOF). For example, its arm is reconfigured by the user through removing or adding DOF, e.g. by removing or adding a link connecting two joints[1]. Another example is the transfer to another robot that is supposed to be controlled by the same cognitive architecture. This means that the architecture should *not* be tailored to a particular robot but rather learn to control robots that are different in their number of DOF and different in their size. Learning to coordinate a given number of DOF should happen quickly, taking only a few minutes.

4. **Adaptation to motor deficits:** Some robots may not have high-precision motors. A robot can have joints with motors that suffer from backlash. Small differences in the commanded target positions cannot be resolved, making the joints stand still. Thus, the robot has to be able to adapt to this mechanical property by adjusting the proprioceptive feedback through consecutive target motor positions that cause motion in the joints, i.e. changing the joint input. Related to this, the robot would have to be able to predict how well it would perform a coordination skill, such as reaching. It would have to predict the resulting deviation between the target and its end-effector as outcome of the reaching action.

   In general, the cognitive architecture producing the motor commands has to be able to adapt to different robot platforms, i.e. to robots that have precise motors as well

[1] Note that a robot joint can contain more than one DOF; for instance, a 2 DOF joint contains 2 rotation axes.

as to robots with imprecise motors.

Points 1 and 2 represent adaptation in the *sensory* space, more specific in the *visual* space[2]. The cognitive architecture should enable the robot to adapt to short-term as well as to long-term loss or deficits of visual features that are caused by various reasons, e.g. camera failure or occlusion of the robot's end-effector (Point 1) or occlusion of a target object (Point 2).
Points 3 and 4 represent adaptation in the *motor* space, more specific in the *proprioceptive* space. The cognitive architecture should enable the robot to adapt to changes in the number of DOF (Point 3) and to adapt to motor deficits (Point 4). Note that Points 3 and 4 together are important, since they imply the *support of different robot platforms by one and the same cognitive architecture*. The number of DOF and the type of motors vary from robot to robot. This respectively demands *scalability to the number of DOF* and *adjustment of proprioceptive feedback*.
Thus, the question arises how to built a new cognitive architecture that is aimed at sensory-motor learning and yields such adaptivity and robustness. In particular, what are the key principles and mechanisms it should be composed of, and what are the sensory-motor skills that result from the architecture design? This thesis is aimed at answering these questions.

1.2. Problem Description and Research Questions

In this thesis, the main **problem** addressed is the *autonomous acquisition of sensory-motor coordination skills* for humanoid robots. This problem is settled in the domain of sensory-motor learning in developmental robotics.
This problem is important, since the resultant methods would represent a useful contribution towards open-ended learning [217], [109], [143] that is currently a great challenge in developmental robotics.
My research considers the learning abilities of the human cortex. This implies that a cognitive architecture controlling a robot should not be task-specific but rather general [217]. Furthermore, the architecture has to contain methods that facilitate the learning and incorporation of new knowledge or capabilities over time [143]. Strongly related to learning in developmental agents, a particular challenge is the problem of *poverty of stimulus*[3] [192, p. 260], [193]. At the beginning of operation, embodied developmental agents (biological or artificial) are constrained to poor sensing abilities and domain knowledge. They have to make sense of a small amount of data and still learn enough skills that in turn are useful in the next developmental stages. This is a crucial point that distinguishes a developmental system from a traditional artificial intelligence (AI) system that is fed with a vast amount of data, data collected by an external intelligence than by the system itself.
Elaborating this problem further, the overall aim would be to create a learning system controlling a robot that starts with a very limited amount of capabilities at the beginning of its

[2] Another term for visual space would be *visual modality*.
[3] The term *poverty of stimulus* was originally introduced by Chomsky [45] in the domain of language acquisition but is also relevant for developmental skill acquisition in multiple domains.

operation and acquires skills or capabilities in a step-by-step manner through interaction over multiple developmental stages.
For the scope of this thesis, sensory-motor skills should be acquired by the robot itself through environmental interaction but without domain knowledge provided by a human. This also means that no kinematics (neither forward nor inverse) is provided. Nevertheless, a human can interact with the robot and the robot may learn additional skills by observing meaningful features during the interaction, e.g. by observing the motions of external objects moved by humans.
I propose to tackle the problem by an embodied robotics approach leading to the construction of a technical framework that is a developmental cognitive architecture. In order to create this kind of architecture from a biologically-inspired point of view, I deduced the following **research questions**:

1. What is the key computational mechanism that can be re-used in a multi-purpose manner in order to facilitate sensory-motor skill acquisition?

2. What is the state of the art regarding such a mechanism? What are its limitations?

3. How can those limitations be overcome? What are the solutions in terms of new methods or algorithms?

4. How can these methods be integrated together in order to form a new developmental cognitive architecture? What sensory-motor skills result from the architecture?

1.3. New Approach

I propose to approach the problem of autonomous sensory-motor skill acquisition from several theoretical directions. My approach to the construction of a new developmental cognitive architecture is strongly biologically-inspired and blends together a set of principles and methods originating from AI, neuroscience, and developmental robotics.
The architecture design is grounded on biologically-inspired learning paradigms [55], in particular *supervised learning* and *bootstrap learning*. For a supervised learning method in this context, the teaching data should still be generated autonomously.
I use *predictive coding* [192] to process spatiotemporal patterns. Predictive coding is beneficial, since it supports learning, prediction, and recognition of spatiotemporal patterns through error minimization.
It is important to note that robot skills should be emergent, i.e. they should result from the architecture, instead of being programmed in advance by a task-specific architecture. In order to consider this line of thought, I implement certain methods of *ongoing emergence* [143]. Furthermore, the requirement of autonomy in the process of skill acquisition needs *verification and grounding* [180].

1.4. Contributions

Overall, the main contribution of this thesis is the *self-verifying cognitive architecture (SVCA)* that is a new *developmental cognitive architecture* for the *autonomous acquisition of sensory-motor skills* on humanoid robots. This architecture should add new insights to sensory-motor learning in robots and overcome some limitations of existing architectures and learning methods.

To this end, I have developed new methods, implemented, and validated them separately from each other. The subsequent integration of these methods has led to the new architecture that I have validated as an entire system.

In a bottom-up process, the theoretical development, implementation, and experimental validation yield the following key contributions:

1. In predictive coding, a state-of-the-art method is the multiple timescale recurrent neural network (MTRNN) [230] which can learn sensory-motor sequences in a supervised manner. However, the original MTRNN requires the training of additional neural networks for pre- and postprocessing the input-output data, one network for visual data and another one for proprioceptive (i.e. motor) data. These additional networks are topology preserving maps, also referred to as self-organizing maps [88], [89].
   Instead of training these additional networks prior to the main network, I propose a *modified version of the MTRNN* that does not require additional networks. I replaced softmax activation by sigmoid activation, making the entire network to consist of sigmoid units only. This version of the MTRNN supports *pre- and postprocessing by an analytical method*. Furthermore, I added an *early stopping method* to the training procedure of the network in order to prevent the overfitting of teaching data.

2. Inspired by the structural development of the human cortex, I extended the proposed version of the MTRNN by an evolutionary optimizer. The resulting system is called *evolutionary optimized multiple timescale recurrent neural network (EO-MTRNN)*. Many state-of-the-art methods for spatiotemporal learning use recurrent neural networks where the hyperparameters are often set manually by a human expert who has to conduct several experiments in a trial-and-error manner until suitable hyperparameters are found. In contrast to a manual choice of hyperparameters, I propose the EO-MTRNN that is able to automatically estimate crucial hyperparameters given a set of teaching data. The hyperparameters are the timescales of the neural groups (input/output, fast context, and slow context) as well as the number of neurons per context group. The benefit of this *autonomous hyperparameter estimation (AHE)* is an increase in the learning performance. I also propose a benchmark dataset in order to evaluate the EO-MTRNN by training single as well as multiple sequences simultaneously. Using the benchmark dataset, the EO-MTRNN improved the learning performance compared to a non-optimized version by approximately 43 % in

average without overfitting the given teaching data. Sensory-motor data from a robot were also used as teaching data and the corresponding results show that the EO-MTRNN can also regenerate data that were not part of the training set during the hyperparameter optimization process.

3. State-of-the-art learning methods such as deep learning require a considerable amount of teaching data to function well. However, a developmental system[4] that learns to purposefully interact with the environment suffers from a poverty of (sensory) stimuli [192, p. 260], [193]. This means that at the beginning of its operation, the developmental system does not have access to a major amount of data. The system is constrained in its ability of perception through its sensors and constrained in its knowledge. This implies that a developmental system has to already make sense of very few data samples in order to generate purposeful or meaningful behaviour.
   For this reason, I propose a new set of methods that together generate meaningful robot behaviour from a very small amount of sensory-motor data samples compared to other existing methods. One of the proposed methods is the *constrained DOF exploration* that uses a simple proportional control algorithm to move each DOF of a particular robot limb within a limited range. During this constrained DOF exploration, sensory-motor features are sampled and used as teaching data representing the ego-motion of the robot. Based on the constrained DOF exploration, I propose biologically-inspired algorithms in order to process self-motion (i.e. ego-motion) as well as motions of external entities, both learned by EO-MTRNN blocks. The proposed biologically-inspired algorithms form a system that is referred to as *predictive action selector (PAS)*.
   Together with the constrained DOF exploration, the proposed PAS overcomes the limit of predictive coding methods [192] that do not support an autonomous acquisition of sensory-motor coordination skills. The PAS can learn basic hand-eye coordination from a minimum amount of samples, without any human intervention. Each sample is a vector containing position[5] as the only physical quantity. For the PAS, approximately 50 sensory-motor samples are enough to acquire object tracking and evading skill by head motions. Furthermore, approximately 150 sensory-motor samples are enough for the PAS to acquire a reaching skill that it can improve later on through a multi-staged operation mode where one stage generates training data for the next one. The improved reaching yields a reduction of the time that the robot needs in order to reach some of the goal positions in its workspace.
   Moreover, the PAS can learn the dynamics of external entities, such as moving objects, through observation. Learning the dynamics is beneficial for predicting the object location, yielding a more robust interaction. The learning and recognition of object trajectories by the PAS yield adaptivity and flexibility, and complement the related work where the dynamics is given beforehand, such as [60, p. 135].

[4] biological or artificial
[5] in the visual space and in the motor space

The PAS is *robust to short-term loss of sensory data* through sensory-motor prediction that continuously generates motor commands, whereas other existing methods (e.g. [83], [159]) stop to move the robot if sensory data are lost.
On the sensory-motor level, the PAS integrates action selection and action generation into one framework. On the sensory-motor level, this overcomes the limitation of motive-driven action selection [209], [211], [206], [207] that de-couples action selection from action generation, at the expense of adaptivity and flexibility.
The proposed PAS has been successfully validated on two different robots (NAO [175] and TOMM [52]), which highlights its cross-platform applicability.

4. Building on the aforementioned contributions, I propose a new *developmental cognitive architecture* — the *SVCA* — for the autonomous acquisition of robot sensory-motor skills. My proposed architecture is a hybrid cognitive architecture with the PAS as its main building block. The architecture complements the PAS modules by an additional short-term memory for visual and motor features in order to facilitate improved coordination, e.g. reaching for an object regardless of head position. Compared to other existing cognitive architectures or robot learning systems, the proposed cognitive architecture is able to bootstrap meaningful behaviour (e.g. reaching) from minimum amount of self-generated data and is able to improve the behaviour over time in order to become more robust to environmental disturbances (e.g. blindfolded reaching). The capabilities of the proposed architecture yield *adaptivity* and *robustness* in the sensory-motor domain.
On the methodical or algorithm level, the majority of the capabilities is realized through *loops of imaginary trial and physical trial (LITAPT)* of actions. These loops allow a *self-verification* of the sensory-motor data during the interaction.
In the sensory (i.e. visual) domain, the autonomous verification of data allows the architecture to self-generate the samples used to train further predictor modules for improving its sensory-motor skill. This yields adaptation and robustness to environmental disturbances. For instance, the architecture can enable a robot to reach goal points in its workspace, even if the robot's end-effector is fully occluded or covered, or even if the robot's cameras fail during the entire interaction. This makes the architecture *robust to long-term loss of sensory (i.e. visual) data*.
In the motor domain, the LITAPT allow an *adaptation to different robot platforms or morphologies* by adjusting the proprioceptive feedback, i.e. the difference between the consecutive target motor positions to cause a changing joint input, in order to compensate for backlash in the joints or motors. An optimal range for the proprioceptive feedback can be found by minimizing the discrepancy between the outcomes of the imaginary trials and physical trials while at the same time avoiding self-collision. This motor adjustment also sets my proposed architecture apart from other existing architectures.

5. A further contribution is an *episodic memory* encoded by a higher-order Hopfield network (HHOP) that processes visual features such as shape and *colour*. Here, I

extended the existing work by adding basic colours (red, green, blue) to the feature space of the HHOP. Although not integrated into the framework of the proposed cognitive architecture, the episodic memory can be considered as its useful extension to trigger actions through sensory-motor association.

1.5. Thesis Outline

Besides this introduction chapter, I present the following chapters in this thesis:

Chapter 2: Related Work. The main part of the related work contains a review of cognitive architectures for sensory-motor learning, as well as related methods or systems. The chapter describes their limitations and mentions open research issues. The theory of predictive coding is also described. The chapter also contains a background section explaining definitions and methods that are the foundation of the theory presented in the subsequent Chapters 3 to 5. The chapter ends by comparing the related work with the new work proposed in this thesis.

Chapter 3: Evolutionary Optimized Multiple Timescale Recurrent Neural Network. This chapter describes the modification and extension of the MTRNN [230]. The original version is modified regarding to the neuron activation function and is extended by an evolutionary optimization method. The resulting system is called EO-MTRNN that can automatically estimate its hyperparameters. In the new architecture, the EO-MTRNN is the basic building block implementing a type of procedural memory. Its purpose is the learning, recognition, and prediction of spatiotemporal patterns.

Chapter 4: Predictive Action Selector. The EO-MTRNN is a spatiotemporal learner that uses a supervised learning method. It requires teaching data. Nevertheless, the teaching data should be generated by the system itself in order to facilitate an autonomous acquisition of behaviour. This chapter explains biologically-inspired algorithms that generate a minimum set of teaching data and use it to bootstrap meaningful behaviour, i.e. sensory-motor skills, on robots with several DOF. The proposed algorithms build on the EO-MTRNN and are integrated into a framework that is referred to as PAS.

Chapter 5: A Self-Verifying Cognitive Architecture. This chapter describes the new cognitive architecture that is the main contribution of this thesis. While the PAS can learn to control a group of DOF, e.g. head joints or arm joints, it is not enough robust when relevant sensory input is missing for a *long* period of time. Additional predictors and mechanisms are needed in order to facilitate a robust interaction and an adaptation to different robot platforms, with capabilities and skills emerging across several developmental stages. Therefore, the PAS is combined with a multi-layered perceptron (MLP) as well as with a state machine and logical rules for verifying the sensory-motor data. Altogether, these components form the new cognitive architecture. Its operating principle is the LITAPT, in which an imaginary action is followed by a physical action, with verifications of each sensory sample as well as verifications of the action outcome. The proposed architecture yields autonomous skill bootstrapping and sensory-motor adaptation capabilities.

Chapter 6: Conclusion. The conclusion chapter provides a final summary of my research in this thesis. The chapter mentions key insights and potential future work.

The contributions that are outlined in Section 1.4 are distributed across the Chapters 3, 4, and 5. Figure 1 shows how these chapters are connected and shows the most important achievements.

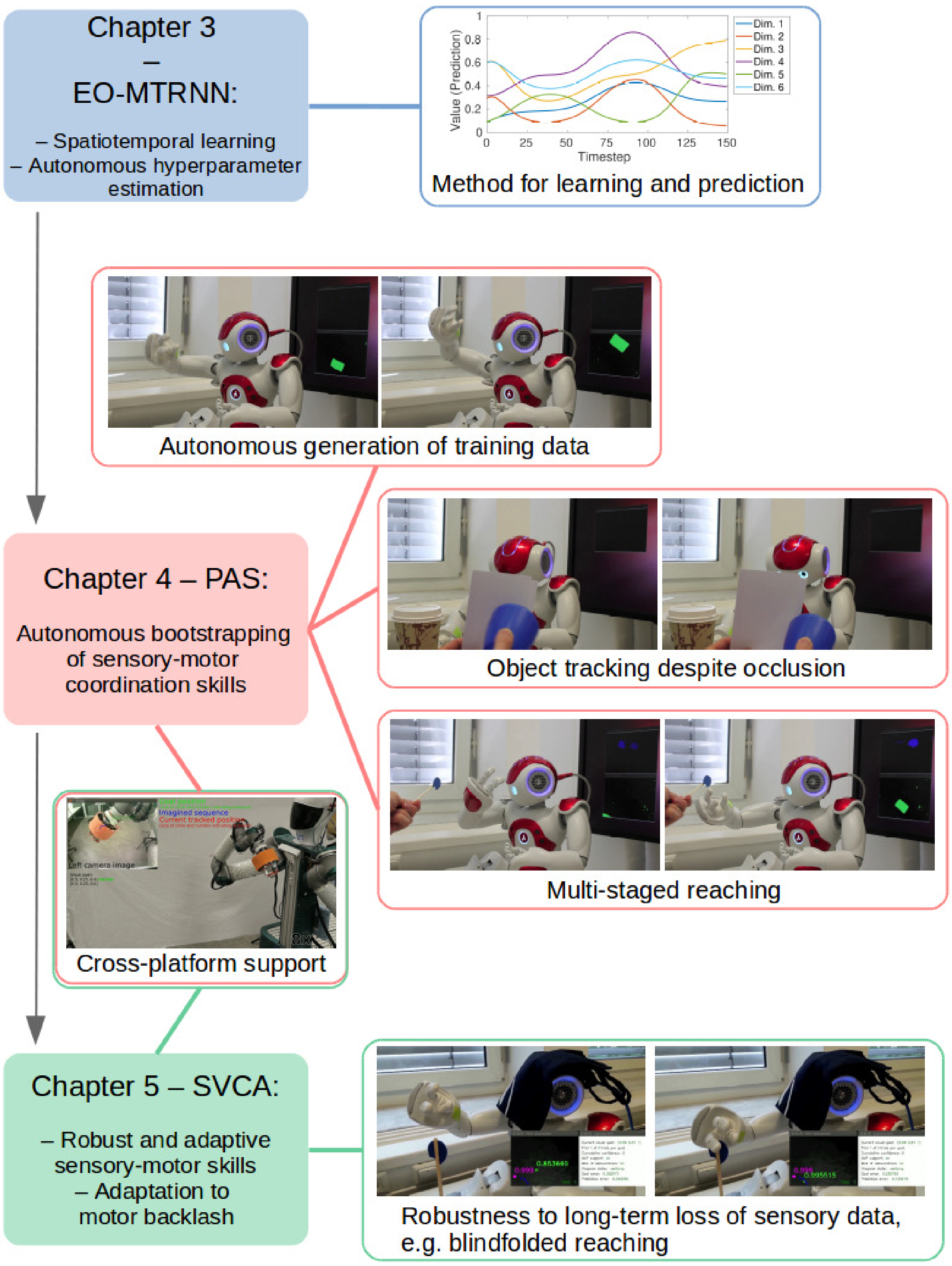

Chapter 3
–
EO-MTRNN:
– Spatiotemporal learning
– Autonomous hyperparameter estimation
Value (Prediction)
Timestep
Dim. 1
Dim. 2
Dim. 3
Dim. 4
Dim. 5
Dim. 6
Method for learning and prediction
Autonomous generation of training data
Chapter 4 – PAS:
Autonomous bootstrapping of sensory-motor coordination skills
Object tracking despite occlusion
Multi-staged reaching
Cross-platform support
Chapter 5 – SVCA:
– Robust and adaptive sensory-motor skills
– Adaptation to motor backlash
Robustness to long-term loss of sensory data, e.g. blindfolded reaching

Figure 1 Overview on the proposed methods in this thesis. The contributions (Section 1.4) are distributed across the Chapters 3, 4, and 5. The proposed EO-MTRNN is part of the proposed PAS that in turn is part of the proposed SVCA. The transparent boxes with images show the key results obtained, in particular on the robots. The robot behaviour is autonomously acquired and reflects the example scenarios given by Points 1, 2, 3, 4 in Section 1.1.

1.6. Publication Note

Located at the beginning of this thesis, the "List of Publications" contains the papers that have been published in peer-reviewed journals and conference proceedings including workshop. The following itemization shows their distribution across particular chapters in this thesis.

Parts of the material in **Chapter 3** have been published in:

- **Erhard Wieser** and Gordon Cheng. EO-MTRNN: Evolutionary optimization of hyperparameters for a neuro-inspired computational model of spatiotemporal learning. *Biological Cybernetics*, 2020. `https://doi.org/10.1007/s00422-020-00828-8`.

Parts of the material in **Chapter 4** have been published in:

- **Erhard Wieser** and Gordon Cheng. Predictive action selector for generating meaningful robot behaviour from minimum amount of samples. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Genoa, Italy, pages 139–145, 2014.
  Video-link: `https://youtu.be/1s1IlVbd444`

- **Erhard Wieser** and Gordon Cheng. Progressive learning of sensory-motor maps through spatiotemporal predictors. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Cergy-Pontoise, Paris, France, pages 43–48, 2016.
  Video-link: `https://youtu.be/fVzIxPxT7MY`

- Wolfgang Burger*, **Erhard Wieser***, Emmanuel Dean-Leon, and Gordon Cheng. A scalable method for multi-stage developmental learning for reaching. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Lisbon, Portugal, pages 60–65, 2017. *Equal contribution to the paper.
  Video-link: `https://youtu.be/okZz9HPRrZI`

Parts of the material in **Chapter 5** have been published in:

- **Erhard Wieser** and Gordon Cheng. A self-verifying cognitive architecture for robust bootstrapping of sensory-motor skills via multipurpose predictors. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):1081–1095, 2018.
  Video-link: `https://youtu.be/T3SgNKPZ3z4`

- **Erhard Wieser** and Gordon Cheng. Forming goal-directed memory for cognitive development. In *Proceedings of Humanoids 2012 Workshop on Developmental Robotics: Can developmental robotics yield human-like cognitive abilities?*, pages 38–39. Workshop at the *IEEE International Conference on Humanoid Robots*, Osaka, Japan, 2012.
  Video-link: `https://youtu.be/XoJHJKzSx2w`

Note that I have provided a media attachment (i.e. a video-link) to most of these papers, demonstrating the obtained results through robot experiments.

2. Chapter

Related Work

The focus of the related work is a review on existing cognitive architectures for sensory-motor learning, including related methods and systems.
Prior to that, the scientific background in developmental robotics and learning is described. I explain existing key principles in this field of research. Then, I summarize the general background in cognitive architectures, before narrowing down the focus on architectures for sensory-motor learning. Finally, I deduce limitations and open research issues.
Thus, this chapter is structured as follows: Section 2.1 briefly introduces the field of developmental robotics. It provides a motivation why learning is important for robots and it summarizes biologically-inspired learning paradigms.
The chapter continues with a description of cross-disciplinary state-of-the-art principles that are relevant for my research:

- Forward and Inverse Models (Section 2.2)
- Learning by self-exploration (Section 2.3)
- Predictive coding (Section 2.4)
- Ongoing emergence (Section 2.5)
- Verification and grounding (Section 2.6)

Section 2.7 provides background knowledge on cognitive architectures and considers architecture design from a developmental point of view related to the principle of predictive coding. Sections 2.8 and 2.9 set the focus of this chapter and describe technical systems that are related to some of the aforementioned principles exploited for their design. In particular, Section 2.8 describes and compares state-of-the-art *mechanisms for action selection*. Section 2.9 describes state-of-the-art *cognitive architectures for sensory-motor learning* and provides *comparison tables* that identify key characteristics of existing architectures for sensory-motor learning.
Finally, Section 2.10 deduces the *limitations* of state-of-the-art systems and mentions *open research issues*.

2.1. Scientific Background I — Developmental Robotics and Learning

Developmental robotics emerged as a cross-disciplinary field from robotics, developmental psychology, neuroscience, and AI. Major proponents of this field have been Weng et al. [217], [215], Asada et al. [9], [8], and Lungarella et al. [109]. In the following, I briefly summarize the origins of developmental robotics, its key idea of mental development, and biologically-inspired learning paradigms.

2.1.1. Behaviour-Based Robotics

Developmental robotics builds on the earlier ideas of embodiment and situatedness[1] proposed by Brooks [30], [31], and then further elaborated by Pfeifer and colleagues [140], [138]. Embodiment requires that the agent has a morphological structure with sensors and actuators connecting it with the real world [137]. Situatedness means that the world is seen from the agent's perspective [137]. In sum, embodiment and situatedness base on the physical *grounding* hypothesis that is in strong contrast to the physical *symbol system* hypothesis [124] originating from the early days of AI. The physical grounding hypothesis states that the world, which the robot is embedded into, is its own best model containing every relevant detail [30]. This implies that a robot must be equipped with appropriate sensors and actuators. Sensor data are tightly coupled with actions. The actions form the robot's behaviour reflecting the robot's goals. Technical instantiations of the physical grounding approach, e.g. the subsumption architecture [29], are often assigned to the category *behaviour-based robotics* [110], [111]. The tight coupling between sensations and actions is inspired by evolutionary biology where its is essential for a biological agent to move around in a dynamic environment and to sense the environment to properly (re)act in order to survive. This concept can be transferred to an artificial (robot) agent. For example, an inspection robot moves along a railway track, carefully inspecting the current track section until it quickly leaves the track when it senses an approaching train [29]. Although the behaviour-based approach has led to robots that can act quickly and robustly in dynamic environments, it is still limited to low-level reactive capabilities. Learning and higher-level cognition are not considered.

2.1.2. Autonomous Mental Development

Weng et al. [217] proposed autonomous mental development for robots in order to overcome the limitations of existing AI systems that were designed to be task-centric, fed with human-labelled data, or evolving but only in virtual (simplified) worlds. In [217], the main idea behind a developmental robot is that its body is designed according to the ecological working conditions (i.e. its environment) and that the robot is controlled by a developmental program. The developmental program enables the robot to interact with the world and a human over longer periods of time. Through environmental and social interaction, the developmental program learns stage-by-stage new capabilities, skills, and knowledge. As outlined in [217]

[1] Situatedness is also referred to as *situated activity*.

and [215], the key difference to a traditional robot program is that a developmental program is *not* task-specific but rather general. This means that the developmental program can create representations of a newly learned task or capability and that it can learn in an open-ended manner.

2.1.3. Biologically-Inspired Learning Paradigms

For a robot, one may ask the question why learning matters at all? There are many reasons. One reason fits well with the aforementioned idea of embodiment. For a humanoid robot, learning helps to gather information about its body and its environment, which are both too complex to be modelled. Even if the kinematics and dynamics of the body are known, a real sensory input often differs from the one derived by an *a priori* model because the sensory input is always influenced by the interaction between robot body and environment. For example, when the robot grasps an unknown object, the physical parameters of its arm (e.g. mass and momentum) differ from the nominal state depending on the grasped object. It is cumbersome to estimate all possible state variations in advance because of unknown data distributions and noise. Here, the deployment of appropriate learning methods is beneficial, since these methods enable the robot to explore the environment and to extract information in order to build an internal model of its body and environment.
Many learning methods are based on insights from neuroscience and cognitive science. Especially neuroscience offers guiding principles for the development of cognitive robots which can learn and adapt. One can distinguish three major types of learning and relate each of them to particular areas of the human brain [55]: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. All these types establish a mapping between input and output data, but each of these types has its own particular method of learning this mapping. Often, the input and output data are multi-dimensional and the mapping is non-linear. Each of these types of learning can be mapped to particular regions of the human brain.

2.1.3.1 Supervised Learning

Supervised learning uses error signals to learn the mapping. Error signals result from the discrepancy between the desired output data[2] and the actual output data generated by the system. This implies that the desired data are made available to the system. Often, the desired data are provided by a human teacher (supervisor). This type of learning is related to the learning that takes place in the cerebellum [55].

2.1.3.2 Unsupervised Learning

Unsupervised learning acquires the mapping between input and output data in a self-organized manner without any external signals, e.g. without an error signal, to evaluate a possible output data. Processing is performed based on the input data only. Unsupervised learning is bene-

[2] The desired output data is also referred to as target data or teaching data.

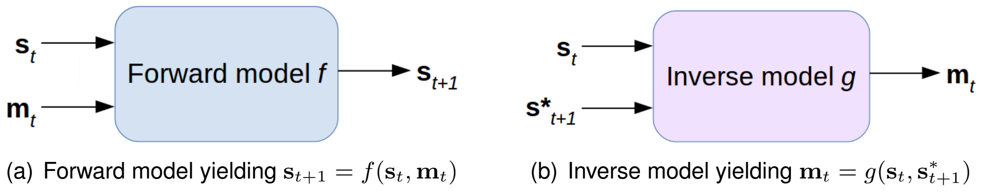
ficial when the goal is to find hidden structure in unlabelled data. Doya [55] hypothesizes that the neocortex learns in an unsupervised manner. Nevertheless, there is increasing evidence suggesting that the neocortex also deals with error signals when comparing the predictions generated in higher-level cortical areas with the incoming sensory signals from lower-level areas [63], [6]. Thus, in addition to unsupervised learning, a form of supervised learning might also take place in the cortex.

2.1.3.3 Reinforcement Learning

Reinforcement learning [187] uses a scalar reward signal to learn the mapping. Often, this type of learning is based on trial and error. The system obtains a positive reward for a successful action outcome, and a negative reward for an unsuccessful outcome. Thus, correct mappings can be learned by reinforcing them, maximizing the cumulative reward. Reinforcement learning is suitable when an input-output mapping does not exist at the beginning of operation. For example, when the mapping from sensory states to actions is not known, but a method for evaluation of a given input-output sample exists instead. Reinforcement leaning is related to the learning that takes place in the basal ganglia [55].

2.2. Forward and Inverse Models

Jordan and Rumelhart [81] introduced forward and inverse models for the processing of sensory-motor data. Figure 2 shows the model input and output. In sum, a forward model f



(a) Forward model yielding $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{m}_t)$ (b) Inverse model yielding $\mathbf{m}_t = g(\mathbf{s}_t, \mathbf{s}^*_{t+1})$

Figure 2 Forward model and inverse model for processing sensory data $\mathbf{s}$ and motor data $\mathbf{m}$.

takes the current sensory sample vector $\mathbf{s}_t$ (sensory state) and motor sample vector $\mathbf{m}_t$ (motor state) as its input, and then delivers the predicted sensory state $\mathbf{s}_{t+1}$ as its output, formally $\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{m}_t)$. The prediction $\mathbf{s}_{t+1}$ can be interpreted as the predicted sensory outcome that is caused by $\mathbf{m}_t$ given $\mathbf{s}_t$.

An inverse model g takes the current sensory state $\mathbf{s}_t$ and the desired sensory state $\mathbf{s}^*_{t+1}$ as its input, and then predicts a motor command $\mathbf{m}_t$ as output, formally $\mathbf{m}_t = g(\mathbf{s}_t, \mathbf{s}^*_{t+1})$. Given the current sensory state $\mathbf{s}_t$, the computed motor command $\mathbf{m}_t$ should achieve the desired sensory state $\mathbf{s}^*$ at the next time step.

Multiple forward and inverse models can be connected together to form pairs of coupled forward-inverse models [227]. This allows a context-based selection among controller models.

2.3. Learning by Self-Exploration

Learning by self-exploration enables the robot to autonomously develop an internal model of its body and environment. The acquired internal model may contain representations of the robot's kinematics and body, and/or representations of object interaction. Learning by self-exploration can use a mix of the biologically-learning paradigms (Section 2.1.3).
The learning of the kinematics including related body representations can be divided into a motor babbling approach and a goal babbling approach. The motor babbling approach acquires the internal model by exploring the motor space[3] through random motion of the joints. Example work comprises visuo-motor learning to identify body parts and tool extensions [151] and cross-modal learning with action recognition [152].
The goal babbling approach [148] acquires the model by a path-based sampling method where training data are collected along paths resulting from execution of the currently learned estimate along a desired path towards a goal. Example work [18] shows that for the learning of inverse models (Section 2.2), an exploration of goals in the task space is faster than an exploration in the motor space.
The common principle of babbling is that during robot motion, sensory data from other modalities such as vision (e.g. [159]) and touch (e.g. [61], [115]) are taken into account, and then correlations between multiple sensory modalities can be learned and exploited.
Self-exploration can also involve an object which the robot interacts with. For example, the robot can slightly push the object and then evaluate the perceptual state after it has performed the pushing action. Thus, an initial perceptual state, a robot action, and the following perceptual state (representing the result of the action) can be brought into a relationship. This is known as object-action complexes [228] and affordances [203]. A robot can further use its visuo-motor experience in order to acquire a generative model for identifying its own body parts and external objects from unlabelled sensory-motor sequence [128]. The identification of own body parts and their distinction from other entities is also known as self-perception, which can combine visual, proprioceptive, and tactile cues [101].
In the following, I describe some example work on learning by self-exploration.

2.3.1. Selected Examples

Saegusa et al. [151], [152] propose a system for the exploratory learning of self-other distinction by self-motions and the exploratory learning of the effects of actions applied to objects.
In [151], the robot moves its arm while observing the proprioceptive and visual feedback. If the feedback is correlated, the robot will define the perceived object as its own arm. Based on this self-exploration, the robot can predict the appearance and location of its arm.
In [152], the robot acquires own body perception, fixation, reaching and grasping, predictive human action recognition, and cross-modal prediction of sensory effects. Minimal knowledge has been provided to the system beforehand in order to establish a rich motor exploration.

Baranes and Oudeyer [18] propose an intrinsically-motivated goal exploration system that

[3] The motor space is also referred to as joint space.

can learn inverse models in a highly efficient manner. Their system is based on their earlier work [17]. They combined goal babbling and intrinsically-motivated learning, the latter suggested in [132]. The combination allowed a robot arm to learn its limits in the reachable space.

Ugur et al. [203] propose a system for exploratory learning of object affordances. According to [203], an object affordance is the relation between an action (or behaviour) performed on a particular object and the corresponding effect of that action. The approach is motivated by infant development. Infants learn the dynamics of objects by observing the effects of their actions applied to the objects. Similarly, the system [203] learns object-action-effects. In the first step, the system discovers effect categories in an unsupervised manner. In the second step, the system learns the mapping between object features and effect categories. The benefit of learning the effect categories is that the robot can predict whether an action performed on an object would accomplish a desired environmental change. After learning, the system is able to plan its actions in a goal-directed manner in the perceptual space.
Building on their previous work, Ugur et al. [204] propose a developmental framework that enables a robot to discover motor primitives like *grasp*, *hit*, *carry*, and *drop* in an unsupervised manner. Their system emulates a part of developmental progression in infants. At the beginning, the robot's action repertoire consists of only one basic action (*swing hand*) and one reflex (*grasp*). The robot discovers meaningful primitives by executing its basic action at different hand speeds and observing the corresponding tactile sensations. When focusing on the *grasp* primitive, the robot learns the affordances of several objects by using the algorithms from [203].

Reinforcement learning partly overlaps with self-exploration. The agent explores possible actions on its own and receives reward signals. An example is the work by Nassour et al. [120]. They propose an algorithm that learns a task through stages of trials, evaluations, and decisions. The algorithm is neurobiologically-inspired. It models the interaction between the anterior cingulate cortex and the orbitofrontal cortex of the brain, central pattern generators in the spinal cord, and the environmental feedback on the body. In [120], Nassour et al. use self-organizing map (SOM) [88], [89] and they propose a qualitative adaptive reward for success-failure learning. The qualitative adaptive reward is a part of the equation of SOM weight updates. It can scale the quality of each trial depending on the quality of previous trials. This helps to optimize a task in addition to the learning. Their system is applied to a humanoid robot that learns to walk on different sloped terrains.

Table 1 summarizes and compares these examples on learning by self-exploration.

2.3.2. Advantages and Drawbacks

Overall, the advantage of self-exploration is its autonomy. The robot can acquire an internal model without an external teacher or coach. Cross-modal sensory association further increases the robustness of the acquired model.
The disadvantage of self-exploration is the size of the motor space, or more general, the pos-

Work	**Robot platform**	**Key method(s)**	**Robot skill(s) per method**
Saegusa et al. [151]	iCub humanoid	Correlation of vision and motor signals through motor babbling	Body identification
Saegusa et al. [152]	iCub humanoid	Correlation of vision and motor signals through motor babbling	Body identification
		K-means clustering of sensory features and Bayesian estimation	Action recognition
		Look-up table	Cross-modal sensory association
Baranes and Oudeyer [18]	Redundant arm (8 DOF real, 15 DOF simulated), 4-legged robot (simulated), 4 DOF arm with fishing rod (simulated)	Intrinsically-motivated goal exploration	Learning coordinated control (inverse kinematics) and locomotion
Ugur et al. [203]	Anthropomorphic robot arm and a range camera	Hierarchical clustering by X-means and support vector machine	Learning and prediction of action effect categories
		Tree generation and forward chaining	Goal-directed action planning
Ugur et al. [204]	Anthropomorphic robot arm and a range camera	Evaluating the effect of parameter variation, e.g. speed	Discovery of motor primitives
Nassour et al. [120]	NAO humanoid robot	SOM	Gathering of success and failure experiences
		CPG	Locomotion

Table 1 Example works about learning by self-exploration.

sible spaces being explored over time. For example, humanoid robots have a large number of DOF, which increases the search in the motor space drastically. Although goal babbling reduces the search problem, self-exploration reaches its limits when higher-level tasks should be learned, such as cleaning a table from clutter. That limit can be overcome by imitation learning [95], [38].
Another limitation is the lack of predictive capability. The essence of spatiotemporal prediction, which facilitates many sensory-motor skills in biological agents, is neglected in these works on self-exploration.

2.4. Predictive Coding

The principle of predictive coding has its origins in signal processing [117], where it has been used for data compression [168]. The main idea behind predictive coding is that only the unexpected information should be encoded. A simple example is the deviation between the actual signal and the predicted one. This deviation is referred to as prediction error. Many transmissions carry only the prediction error, resulting in savings on bandwidth.
In this section, I outline the principle of predictive coding by summarizing neuroscientific insights, by making relations to the aforementioned forward and inverse models, and by describing its application in technical frameworks controlling robots. A general overview on predictive coding can be found in [50].

2.4.1. Message Passing and Temporal Hierarchies in the Cortex

According to neuroscientific theories [59], [60], [1], [84], the human cortex implements predictive coding by a message passing between neuronal hierarchies[4] containing two directions opposite to each other: a top-down (descending) path that is the message passing from higher level neuronal areas to lower level areas, and a bottom-up (ascending) path that is the message passing from lower level neuronal areas to higher level areas. The top-down path transmits predictions, whereas the bottom-up path transmits prediction errors. Work [84] states that the prediction error is the difference between (ascending) sensory input and (descending) predictions of that input. The cortex tries to minimize the prediction errors by updating its beliefs or expectations in the areas that send predictions. The belief update happens through purposefully acting in the world [60], suggesting that the primary purpose of the brain is the generation of actions [226]. Furthermore, there is evidence [90] suggesting that predictive coding in the cortex is not restricted to sensory-motor processing but is also involved in more abstract (higher-level) processing, for example processing that yields abstract skills like theory of mind.
In addition to the cortical message passing, it is important to consider the temporal hierarchies of the signals transmitted between the cortical areas. There is evidence [14], [16] showing that rostral areas that are associated with higher level processing (e.g. planning) have a different dynamics of neuronal activation than caudal areas associated with lower level processing.

[4] Neuronal hierarchies implement a *hierarchical generative model* [1].

A keyword here is *temporal abstraction* [16], meaning that the more abstract the goal of an action is, the longer is the timescale of the governed actions. For example, the goal of preparing a cocktail is relevant for longer time than the goal of slicing a lime needed as ingredient. Further example for temporal abstraction is the interaction between the prefrontal cortex, the supplementary motor area, and the inferior parietal lobe [6]. In [6], it is stated that the prefrontal cortex is involved in action planning and once it has found the (optimal) motor program, it triggers initial activation states in the premotor and supplementary motor areas, especially in the ventral premotor area. In [6], it is further stated that the neuronal activity in the premotor and supplementary motor area has a slow dynamics and may represent goal-related abstract sequencing of action primitives, while the inferior parietal lobe has a fast dynamics and implements a sensory forward model. In [196], it is outlined that the inferior parietal lobe uses the goal information input from ventral premotor area in order to predict sensory-motor flow by a forward computation. The recognition of the goal of an observed physical action is accomplished by an inverse computation. The neuronal activity in the inferior parietal lobe includes representations of the action primitives and has a faster dynamics than the activity in the rostral (higher-level) areas. Note that a temporal hierarchy may not only be present between different cortical areas along the rostral-caudal axis but also within one particular area itself such as the prefrontal cortex [15], [169]. It is difficult to exactly determine the temporal hierarchies, since some cortical areas overlap, for example the prefrontal cortex and the premotor area.

2.4.2. Relation to Forward and Inverse Models

The prediction error is a key signal that can also be part of the aforementioned internal models (Section 2.2), for example in [227]. Thus, forward and inverse models should not be considered separately from the predictive coding principle but rather in strong relation to it. In detail, this relation is explained in [1], which relates the *optimal motor control theory* [163] containing forward and inverse models to the *active inference theory* [59], [60] containing a hierarchical generative model[5]. Both theories each optimize the motor commands. The optimal motor control uses state estimates (e.g. position, velocity, force) in order to optimize motor commands. Active inference uses prediction errors in order to generate proprioceptive predictions that are then converted into muscle torques. Further differences between the two theories are out of scope. Details can be found in [1].

2.4.3. Computational Models in Robotics

For computational models of predictive coding, two main directions exist: deterministic and probabilistic. A deterministic model minimizes the error. A probabilistic model minimizes the free energy [60].

In developmental robotics, the idea of predictive coding is realized through computational models that minimize the prediction error. Controlling a robot, a predictive coding framework achieves the following three capabilities by prediction error minimization: *learning*, *generation*, and *recognition* of (sensory-motor) patterns. This has been demonstrated in various

[5] A hierarchical generative model can also contain a type of forward model generating the top-down predictions.

robot experiments by Tani [192]. It is important to note that in many cases, the patterns are *spatiotemporal* patterns encoding an interaction history, which is in strong contrast to only spatial pattern mapping in behaviour-based robotics (Section 2.1.1).

Similar to the aforementioned theories proposed in neuroscience, a predictive coding framework is characterized by its hierarchical structure, see Figure 3. Independent of the number

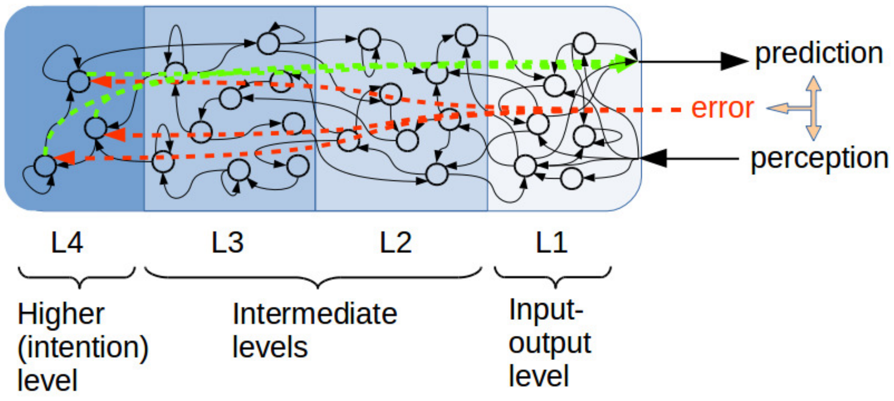


Figure 3 Abstract computational model of predictive coding. The deviation between prediction and perception yields an error signal (red dashed arrows) that propagates from the input-output level across the framework up to the higher (intention) level. There, the error changes the intention that in turn leads to altered prediction (green dashed arrows) passed forward to the lowest level. Note that recurrent neural networks (indicated by circles and their connections) using error propagation methods are beneficial for predictive coding due to their uniform structure that can be scaled to multiple levels of hierarchy.

of different levels of the generative model, the most top level contains representations of intentions of (goal-directed) actions. When an intention gets active, it triggers a stream of predictions that are passed to the lowest level. In the sensory-motor domain, these predictions contain proprioceptive information as well as the resultant perceptual outcome. The proprioceptive information is converted into motor torque generating the action. An error signal arises through the deviation between the prediction of the perceptual outcome and the actual perception. The error signal is propagated back to the highest level and leads there to a modification of intention in case of large errors.

Note that computationally, this interactions between the cortical areas represent a dynamical system where the initial state for a sequence is set at a top (contextual) level. The system dynamics is given by the differential equation $(\dot{\mathbf{x}}, \dot{\mathbf{c}}) = f(\mathbf{x}, \mathbf{c}, \theta)$, where $\mathbf{x}$ denotes the sample in the sensory-motor state space, $\mathbf{c}$ denotes the context state, and θ denotes the model parameters. This dynamics can be expressed by the difference equation $(\mathbf{x}_{t+1}, \mathbf{c}_{t+1}) = f(\mathbf{x}_t, \mathbf{c}_t, \theta)$, where t is the current time step.

Such a dynamical system can be realized by a recurrent neural network (RNN) model. In general, RNN models can have different classes of attractor dynamics, e.g. [116], [127]. Important types of RNN are the Jordan network [79], [80] and the Elman network [57]. Based on the Jordan network, the RNNs by Tani and colleagues [191], [194] realize predictive coding by using parametric bias (PB) neurons as part of their network architecture. PB neurons encode the intention, or goal, of an action that unfolds in time. The initial states of PB neurons are also learned during the training process, in addition to the connective weights[6]. Clusters

[6] Connective weights are also referred to as synaptic weights.

in the space encoded by PB neurons represent classes of different actions, e.g. grasping, pushing, avoiding. Emergent switching of actions can occur by environmental disturbance and by PB modulation.

Yamashita and Tani [230] propose the MTRNN that models the temporal hierarchy in the cortex (Section 2.4.1). Based on the MTRNN, Arie et al. [6] proposed a system that emulates the interaction between the prefrontal cortex, supplementary motor area, and inferior parietal lobule. If one relates MTRNN-based systems to biological brains, then each computational node (i.e. artificial neuron) may represent a cluster or group of approximately hundred thousands of biological neurons.

The aforementioned RNN models are trained by using the backpropagation through time (BPTT) [150], [149] that is a type of supervised learning scheme. In the robot experiments, training happens through kinesthetic teaching where a human teacher guides the robot limbs and shows the robot the actions it should do. In the experiments with the MTRNN, the human sometimes needs to support the robot in switching between the action sequences, since the higher (intention) level is not fully mature yet. The intention level is represented by the initial potential states of the (slow) context neurons and the temporal hierarchy is realized by the different neuronal timescales. The signal flow from the higher-level context neurons to the low-level input-output neurons correspond to the top-down (descending) path, while the backpropagated error signal corresponds to the bottom-up (ascending) path of the theoretical predictive coding model in [59]. MTRNN-based systems can also change the intention in an online manner [44]. For example, an error signal can arise through a behaviour change or a disturbance during environmental interaction that might also include other agents [40]. Then, the error signal propagates through the network within a particular time window in order to change the initial context state and optionally to also modify the connective weights. Changes in the (higher-level) context state in turn alter the signal dynamics in the top-down path, thus influencing the robot behaviour. This phenomenon is also known as *action-perception cycle* where the perception is *not* regarded separately from the action, since both are tightly coupled by mutually influencing each other through time. This action-perception cycle with the restructuring of memory and generation of novel intention is also referred to as *circular causality* [192, p. 238].

Park et al. [135] demonstrate (sensory-motor) sequence learning, learning of motions of external objects under occlusion (i.e. object permanence), and imitation learning, all these capabilities supported by a predictive coding framework using Bayesian methods.

In sum, the computational models of predictive coding encode temporal information that facilitates dynamic sequence learning (circular causality) as a key capability. However, the predictive coding models reach their limits in supporting a gradual (staged) acquisition of capabilities and skills over time, especially in an autonomous manner where the system self-generates its training data.

2.5. Ongoing Emergence

This section summarizes the principle of *ongoing emergence* proposed in the field of developmental robotics[7]. To this end, I first describe some criteria that characterize developmental agents and then I mention similar concepts.

2.5.1. Criteria that Characterize Developmental Agents

Prince et al. [143] suggest the principle of ongoing emergence that they regard as "a core concept in epigenetic robotics". Ongoing emergence addresses the problem of *open-ended skill acquisition*, more generally also referred to as *open-ended learning*. This problem is discussed in other influential works [217], [109], [215], [131] and asks the question how an agent can acquire or learn new skills during its operational lifetime. Obviously, this question involves many sub-questions: What are the operational constraints, i.e. in what environment is the agent embedded into? How does the morphology of an agent looks like? How does the agent's behaviour emerge as result of the interplay between the agent's control mechanisms, its morphology, and its environment [199], [137], [138]? How much *a priori* knowledge does the agent have at the beginning of its operation, i.e. what knowledge and skills have been already provided by the system designer or programmer?

Instead of providing answers to each of these sub-questions, Prince et al. describe six guiding criteria in [143]. Those criteria together form ongoing emergence.

Here, I focus on summarizing three of them, since I regard them as most relevant. These selected three are [143]:

- Bootstrapping of initial skills
- Continuous skill acquisition
- Incorporation of new skills with existing skills

The first point implies that when a developmental agent becomes operational, a first set of skills (also referred to as initial skills) become quickly available through bootstrapping mechanisms. Here, bootstrapping mechanisms refer to any algorithms that facilitate the generation of the initial skills.

The second point means that the agent persists in the generation of skills over its operational time. Multiple approaches exist to realize a continuous skill acquisition. One approach is the dynamic integration of the (control) output of sub-systems, where each sub-system generates a particular behaviour [42], as it is described in [143]. Another approach is the staged accumulation of experience [202], [229].

The third point is difficult to realize because the agent has to determine whether the newly acquired skill is stable, for example by reoccuring with the same perceptual outcome. Moreover, the learning or incorporation of a new skill into the repertoire of skills can cause memory interferences [53]. This is related to the problem of how to integrate a new skill without forget-

[7] Developmental robotics is also referred to as epigenetic robotics.

ting the previous ones.
Note that the principle of ongoing emergence does not depend on the existence of a teacher for showing new skills nor does the principle exclude it. Also, the principle does not suggest any particular technical details. The focus is rather on providing an abstract guideline for the design of agents that show meaningful emergent behaviour.

2.5.2. Similar Concepts

I briefly summarize a few concepts that are very similar to ongoing emergence but have been proposed independently by other researchers under different terms:

- *Scaffolding* [215]: An agent uses existing simple capabilities in order to develop more complex capabilities, with or without a teacher.

- *Incremental process* [109]: An agent builds on its prior structures and functions in order to generate its later structures and functions.

- *Generative learning* [229]: An agent generalizes or transfers its existing knowledge into new domains.

These concepts together help to design emergent systems. However, due to their nature of being a principle or concept, the technical or methodological details are neglected and can be different in the systems realizing these concepts.

2.6. Verification and Grounding

The verification principle and the grounding principle are important because they are regarded as basic principles in developmental robotics [180].
Ayer [10] originally proposed the verification principle that has been later adopted by Sutton [185], [186] in the field of AI. The verification principle basically states that an AI system should not learn anything that it cannot verify for itself [186]. Consequently, the aim of this principle is to let an AI system autonomously verify, i.e. self-verify, the learned knowledge, independent of the domain the knowledge is related to.
For the application of the verification principle to robots interacting with the world, the verification criteria should be determined during the system design phase. The criteria can be expressed by several metrics. For example, one metric can represent the (perceptual) outcome of an action, e.g. successful or not. Such metric is used in robotic systems that are inspired by reinforcement learning in different task domains, for example [120], [142]. Another metric can express the correctness of particular expert controllers and can be applied in relation with reinforcement signals to select between controllers [37].
The grounding principle is stronly connected to verification. Stoytchev [180] explains that the grounding principle constrains the verification, since grounding regulates what to verify and what not to verify anymore. Furthermore, Stoytchev proposes that grounding can be defined

through action-outcome pairs and that the result of an action (i.e. the outcome perceived for verification) must be reproduced several times in the same context. Sensory-motor grounding can also bootstrap the learning of abstract representations, like for example in [93], [69], [128].

2.7. Scientific Background II — Cognitive Architectures

The research field of cognitive architectures has a longer history than the relatively new field of developmental robotics. Similar to developmental robotics, the research on cognitive architectures benefited from multi-disciplinary contributions, ranging from AI, cognitive science, neuroscience, to robotics.
Note that due to the tremendous amount of existing literature on cognition and cognitive architectures, I only extract those aspects that are relevant for my research in this thesis. I provide a general background on cognitive architectures in Appendix A and list some state-of-the-art architectures in Appendix B.
This section briefly describes the following important aspects: definitions (Section 2.7.1), key components common to cognitive architectures (Section 2.7.2), a developmental point of view (Section 2.7.3), and evaluation criteria (Section 2.7.4). These aspects are important because they are directly related to the cognitive architecture that I propose in this thesis.

2.7.1. Definitions: Cognition, Cognitive Architecture, Skill and Capability

In literature, various definitions exist for the same term, depending on the authors' point of view. There is no common agreement how *cognition* is defined. Similarly, expert opinions differ in what a system that is referred to as *cognitive architecture* really is or must contain. In the following, I provide selected definitions that are widely accepted in the research community and I also describe my own point of view.

2.7.1.1 Cognition

Cognition (1), originally defined by proponents of *cognitivism*[8], quoted from [210, p. 153]:

> "(...) computations defined over internal representations *qua* knowledge, in a process whereby information about the world is abstracted by perception, and represented using some appropriate symbolic data-structure, reasoned about, and then used to plan and act in the world."

The aforementioned definition contrasts with biologists' or neuroscientists' thinking about cognition in living systems. An example is the following definition:

[8] Cognitivism is dated back to the thinking in the early days of AI, when intelligence was thought to be captured by symbols that are processed through logical rules.

Cognition (2), originally defined by Maturana and Varela [113], quoted from [210, p. 155]:

> "(...) the process whereby an autonomous system becomes viable and effective in its environment. It does so through a process of self-organization (...)."

Based on Maturana's and Varela's thoughts [113], Vernon [206] extended the definition of cognition.

Cognition (3) quoted from [206, p. 90]:

> "(...) process by which an autonomous self-governing agent acts effectively in the world in which it is embedded, (...) the dual purpose of cognition is to increase the agent's repertoire of effective actions and its power to anticipate the need for and outcome of future actions, (...) development plays an essential role in the realization of these cognitive capabilities."

The first definition of cognition focuses on the way how information is abstracted, structured, and processed internally. It represents the thinking in traditional AI. The second and third definition imply that a cognitive system is embodied and closely interacts with its environment.
In contrast to the first definition, the second and third definition emphasize the importance of principles of self-organization, system-environment interaction, and development. Thus, the second and third definitions reflect the approaches described in developmental robotics (Section 2.1).
Note that cognition also implies a certain degree of *autonomy*. For the second and third definition of cognition, autonomy means goal-directed action selection and execution without human intervention, where the main purpose of the actions is to maintain the existence of the agent [199], [29], [137], [25], [210], [206], [207].

From my own point of view, cognition entails processes of perception, motor control, learning, anticipation (prediction), and adaptation. Cognition emerges through the interaction between an embodied system (e.g. a humanoid robot) and the environment[9]. *Learning* is essential for a cognitive system because learning forms internal representations of knowledge through environmental interaction, see Section 2.1.3. Internal representations help to deal with different contexts and environmental changes. *Anticipation* or *prediction* capability is a necessary requirement in order to declare a system *cognitive*. Thus, cognition can be viewed as the complement of perception, since perception deals with immediate timeframes and cognition deals with longer timeframes [154], [210]. Dealing with longer timeframes means to anticipate a possible (perceptual) outcome of a certain action and to consider that when selecting an action. In biological cognitive systems, anticipation is also necessary to compensate for the latencies in the neural processing [206].

[9] The environment can also contain biological and/or artificial agents.

2.7.1.2 Cognitive Architecture

Similar to the definition of *cognition*, the term *cognitive architecture* is also not uniquely defined among researchers. I state two opinions:

Cognitive architecture (1), defined by Sun in [182] and Newell in [123], quoted from [183, p. 342]:

> “(...) a cognitive architecture is the overall, essential structure and process of a domain-generic computational cognitive model, used for a broad, multiple-level, multiple-domain analysis of cognition and behavior.”

Cognitive architecture (2) defined by Vernon et al. in [210, p. 162]:

> “(...) the architecture of the system is equivalent to its phylogenetic configuration: the initial state from which it subsequently develops.”

The first definition is very abstract and tends toward an understanding of the human mind from a psychological point of view.
The second definition is from a biologically-inspired robotics point of view and puts emphasis on development, since it aims to create the minimum amount of mechanisms and components that are enough to enable a robot to adapt to new environmental situations and to generate new types of behaviour (compare also to *autonomous mental development* in Section 2.1.2 and *ongoing emergence* in Section 2.5).

From my own point of view, a cognitive architecture is a technical framework that integrates perceptual, motor, learning, and prediction capabilities in order to generate meaningful robot behaviour or skills. Neuroscientific insights can be used as guiding principles for the design of a biologically-inspired cognitive architecture.

Cognitive architectures can be split into three different types [210], [208]: *cognitivist* architectures, *emergent* architectures, *hybrid* architectures. Each type has its strengths and weaknesses.

2.7.1.3 Skill and Capability

In robotics context, the term *skill* and the term *capability* are often treated as synonyms, especially in the sensory-motor domain.
In this thesis, I adopt the definition by Prince et al. [143] for the term *skill*. According to [143], robot skills encompass:

- Perceptual abilities

- Internal representational schemes

- Behaviours

All these points have to be seen from the developmental perspective.
In this thesis, I focus on the second and especially the third point. Internal representational schemes are also referred to as *internal representations*. The developmental perspective requires that internal representations are acquired by the agent itself, see also *situatedness* (Section 2.1.1).
Here, the term *behaviour* includes actions. I use the term *meaningful* behaviour if the behaviour serves to accomplish a particular goal in a certain context, i.e. goal-directed actions, or if the behaviour makes sense for a human observer. I also consider a behaviour as meaningful if it serves as a basis for the emergence or development of a (related) subsequent behaviour.

2.7.2. Key Components of Cognitive Architectures

In general, cognitive architectures can be divided into three distinct types [210], [211], [208]: *cognitivist*, *emergent*, and *hybrid*. Cognitivist architectures use symbolic methods and can support many capabilities, but they rely on representations of knowledge provided by the system designer. Emergent architectures rely on principles of self-organization, often using sub-symbolic methods such as artificial neural networks. For robots interacting with the environment, emergent architectures prove to be more adaptive and robust than cognitivist ones, although emergent architectures are (currently) limited in their capabilities. Hybrid architectures combine the strengths of both design paradigms, cognitivist and emergent, by combining symbolic as well as sub-symbolic methods.
Despite these various types, Langley et al. [100], [99] identify some key aspects or components that all types of cognitive architectures have in common:

- *Memories* to store knowledge, beliefs, and goals

- *Representation of elements* in the memories

- *Functional processes* that operate on the memory content

Note that the term *memory* or *memories* refers to an abstract component that contains data structures representing the (gathered) knowledge or the interaction experience of the agent, e.g. a history of sensory-motor states. Memory is important, since it facilitates the *capability of prediction* which is crucial in order to call a system or an architecture *cognitive* [210], [24]. Example architectures with an emphasis on memory can be found in Appendix C.
In the above itemization, the first two aspects are intertwined: knowledge, beliefs, and goals have a certain type of representation depending on the design paradigm, i.e. the representation can be either accessible to direct human interpretation (*cognitivist* design) or not accessible (*emergent* design).
Consider the following example: A cognitivist architecture may use semantic representations

of kitchen utensils and tools, e.g. for a cooking task. These representations are directly accessible to a human expert and often provided by him beforehand. For example, the human can check when a particular node of the semantic network is active at what time and what the node represents. In contrast, the representations in an emergent architecture cannot be understood so easily, since they are often encoded by sub-symbolic methods. For example, a (deep) neural network may contain a distributed encoding of abstract concepts in several neuronal layers. The concepts are then expressed by a series of numbers in high-dimensional vectors that are not interpretable by a human.
The type of representation influences the choice and design of functional processes. The functional processes are methods or algorithms operating on the memory content. Note that knowledge, beliefs, and goals are *not* considered to be part of a cognitive architecture because these memory contents can change over time [100]. According to [100], the architecture is formed by the technical *infrastructure* — i.e. memories (short-term, long-term), representation schemas, and functional processes — that remains constant over time.

2.7.3. The Developmental Point of View — The Significance of Internal Representations Acquired through Interaction

The notion of development should bring robotic and AI systems into a new direction. I have outlined different learning paradigms (Sections 2.1.2, 2.1.3, and 2.3) providing methods that can alter the architecture memory contents in order to acquire new knowledge, to adapt to a new situation, and to generate new robot behaviour as well as to improve it.
In my opinion, the main problem with cognitivist cognitive architectures is that they are 'spoon-fed' by knowledge from a human expert and they cannot give meaning to that knowledge (symbol-grounding problem [216]). The emergent design paradigm (Appendix A.2) attempts to offer a solution this problem, for example by proposing a grounding in the agent's sensory-motor coordination [137]. This view is substantiated by developmental studies, for example Lockman [108], and Thelen and Smith [197].
I suggest that one particular solution is the application of the predictive coding principle (Section 2.4), since that principle can acquire internal representations of abstract and non-abstract entities through the robot's own experience by acting in the world. This is accomplished by self-organization of the neuronal activity in the higher (intention) level as result of the prediction error minimization process. The activity in the higher neuronal level represents the experience of the robot, experience that was acquired and generated by the system itself[10]. Examples of this experience acquisition can be found in [192].

2.7.4. Evaluation Criteria for Cognitive Architectures

Like any other technical system, a cognitive architecture has to be evaluated. Although Langley et al. [100] propose some criteria for evaluating cognitive architectures, their criteria seem to be directed to evaluate cognitivist architectures that may already capture cross-domain knowledge from the beginning of their operation.
Nevertheless, the evaluation of a cognitive architecture is different from the evaluation of a

[10]Note that the robot's interaction may still include a human teacher helping the robot, e.g. through kinesthetic teaching.

task-specific system [137]. In case of a cognitive system, the performance (i.e. achievements over time in a particular task) is not the only issue what counts, since an important aspect is also the understanding of intelligence [137], [140]. This is especially related to emergent or hybrid architectures that model aspects of cognitive development.
For a cognitive architecture realizing a developmental system, criteria should be derived that measure aspects of cognitive development. For example, what skills have been learned or acquired over time, given the amount of *a priori* knowledge? Also, can the architecture improve in a skill over time? Thus, for developmental systems, the evaluation criteria depend on the aspects of development [109], [211] that an architecture is aimed to model.

2.8. Action Selection — A Key Mechanism for Cognitive Architectures

The background on developmental robotics (Section 2.1), a majority of the principles (Sections 2.2–2.6), and the background on cognitive architectures (Section 2.7) together set the foundation for the understanding of the different action selection mechanisms that I outline in this section by describing example works.
Action selection is important because it is a key mechanism for generating robot behaviour. The robot's actions depend on many factors, for example goals or motives, environmental situation, and energy level (for mobile robots). The corresponding action selection mechanisms have to take these factors into account.
I describe example architectures for emergent robot behaviour by focusing on architectural components responsible for action selection and the way these components are connected or integrated. For each type of action selection mechanism, the benefit and the limitation are pointed out. I refer to the original papers for technical details. At the end of this section, I compare the different action selection mechanisms with each other.
Note that according to the component requirements proposed by Langley et al. [100] (see Section 2.7.2), some of the presented architectures may not be called *cognitive* architecture due to the lack of memory components. Nevertheless, as pointed out earlier, a unique definition of cognitive architecture does not exist (Section 2.7.1). The realization of various robot behaviour does not request a control architecture to contain memory. Though, memory is necessary for realizing context-driven action selection, i.e. action selection that depends on the history of states.

2.8.1. Implicit Action Selection in Behaviour-Based Systems

Behaviour-based systems, or autonomous agent robotics [210], realize the interaction of several layers of sub-systems. They have implicit mechanisms for action selection. *Implicit action selection* means that an isolated module selecting the actions does not exist. Instead, the actions emerge through the interaction of many modules. Some prominent examples are proposed in [29] and [110]. Maes [111] proposes the behaviour-based systems approach for designing autonomous agents.

Behaviour-based systems have their origin in the subsumption architecture [29], which introduces several “levels of competence”. In [29, p. 16], a “level of competence” is defined as “an informal specification of a desired class of behaviors for a robot over all environments it will encounter”. The lowest level of competence is “avoid objects”, the next level is “wander”, followed by higher level behaviours such as “explore”, “build maps”, up to “reason about behavior of objects”. The levels are implemented by several control layers. The lowest level of competence corresponds to the lowest control layer. Then, each next level of competence is achieved by connecting a new control layer to the previous one(s) and running them simultaneously. Each control layer consists of many modules (also called processors) connected together. Each module is implemented by an augmented finite state machine providing very simple functionality. Each module has a suppression mechanism (suppressor) at its input and an inhibition mechanism (inhibitor) at its output. When suppression is active, the usual input is suppressed and replaced by another signal fed into the module. When inhibition is active, there are no output signals for a certain time.
In the paper about situated agents [110], a dynamic action selection principle is introduced for agents with goals. A network structure is formed by “competence modules”, where each module carries out a specified action. A dynamic flow of activation energy between the modules realizes emergent action selection; thus, the selection is based on an activation/inhibition dynamics.

2.8.1.1 Benefit

The benefit of the control layers with suppressors and inhibitors [29] is that they support a higher level behaviour (e.g. “wander around”) while simultaneously allowing low level behaviour (e.g. “avoid objects”), resulting in e.g. “wander around aimlessly without hitting obstacles” [29, p. 19]. A prerequisite is a rich sensory interface.
The benefit of the activation/inhibition dynamics [110] is that the action selection can be tuned by parameters, supporting adaptivity to environmental and task features.

2.8.1.2 Limitation

Many behaviour-based systems, such as the subsumption architecture, suffer from scalability problems. A high number of different modules have to be connected together when one wants to design a new control layer allowing a new behaviour, since learning mechanisms do not exist (at least in the original approach). Moreover, it is difficult to establish higher level behaviour, since the subsumption architecture and related behaviour-based architectures are purely reactive systems [49], [210].
Memory is not part of the original behaviour-based systems. Thus, learning is neglected.

2.8.2. Integrative Action Selection

Cheng et al. [42] propose an integrated system design based on their earlier work [41]. In both works [42], [41], they introduce a component referred to as "basic integrator". Many of these basic integrators are used throughout their architecture.

2.8.2.1 Benefit

This concept overcomes the drawback of systems where only one particular module can take control over the robot at a time, e.g. by a winner-take-all mechanism.
The basic integrator establishes sensory-motor connections in an dynamical way by considering confidence and priority of different (sensory) inputs. Thus, each sub-system can participate in the generation of motor commands, leading to emergent human-like responses on a humanoid robot. This is particularly important for a continuous generation of skills [143]. Moreover, the integrators yield adaptive systems because they can provide redundancy in case of failing components. Another benefit is the flexible system design because the integrator concept permits the addition of other components or sub-systems in a simple way.

2.8.2.2 Limitation

Similar to the behaviour-based systems, memory components are not present in the system of basic integrators. Learning is neglected.

2.8.3. Motive-Driven Action Selection

The best example for motive-driven action selection is the *iCub cognitive architecture* [209], [211], [206], [207], which is a state-of-the-art architecture for a humanoid robot.
It belongs to the emergent cognitive architectures. The *iCub cognitive architecture* has a set of innate behaviours. These innate behaviours are basic sensory-motor skills that were inspired by the basic perception-action skills of human infants. The *iCub cognitive architecture* has an explicit action selection mechanism, i.e. this architecture contains a separate isolated module for the function of action selection. The "action selection" module requires input from another module called "affective state". Thus, the selection is guided by affects or motives. In the *iCub cognitive architecture*, the affective state provides motives, i.e. levels of curiosity and experimentation, both levels represented by a temporal series of spikes. The action selection module determines an action as a function of these levels. Since the iCub architecture controls the iCub humanoid robot with a high number of degrees of freedom, the actions are split into eye gaze motions, reaching and grasping, and locomotion. A control module exists for each "gaze control", "reaching and grasping", and "locomotion". Note that in contrast to behaviour-based architectures like [29], [110], and in contrast to [42], the *iCub cognitive architecture* contains memory systems, divided into "procedural memory" and "episodic memory". Consequently, the architecture can gather experience through processes of learning. The output of the action selection module, i.e. the selected action, is fed into the reaching and

grasping module, and locomotion module but also into the procedural memory module. The procedural memory contains associations between representations of perceptual and action states, and it can infer the perceptual state given a particular action.

2.8.3.1 Benefit

The benefit of the procedural memory is its ability to predict perception-action cycles, comparable to a type of forward model (Section 2.2). This prediction in turn influences the action selection by modulating the levels of curiosity or experimentation. Note that the predictions delivered by the procedural memory do not determine the action itself, since they only influence the levels of motives within the affective state module. The benefit of this motive-driven action selection is an increase of the robot's autonomy [207] in exploring the world.

2.8.3.2 Limitation

Many developmental results of the iCub cognitive architecture have not been shown or clearly pointed out yet in experiments. As mentioned in [211], the (technical) realization of this architecture is a long-term project.
The action selection is separated from the action generation itself in the iCub architecture, since *action selection* and e.g. *reach and grasp* are implemented as separate modules. Thus, it is not clear whether the task-space reaching controller (reach and grasp module) can adapt to a different morphology other than the iCub robot's morphology. In other words, it is not clear whether such a de-coupling between action selection and action generation supports adaptivity to unforeseen changes in the robot's morphology, e.g. occlusion of visual features or reduction of available DOF due to damage.

2.8.4. Predictive Coding-Based Action Selection

One approach is to establish action selection by a separate module that switches between motor or action primitives. Action primitives can also be realized by separate modules. When a sequence consisting of different action primitives should be executed, the action selection module has to switch between the outputs of the modules implementing the action primitives. However, one cannot simply increase the number of modules in an attempt to create different new actions, especially when (learned) sensory-motor sequences are similar and overlap with each other [195], [230]. For this reason, Yamashita and Tani [230] propose a MTRNN which can learn overlapping sensory-motor sequences with branching points. The MTRNN is a continuous time recurrent neural network (CTRNN) with at least two different timescales for the context neurons, thus dividing the context neurons into fast context and slow context. The switching between action primitives happens within the neural network itself by changes in the activity of slow context neurons. Slow context neurons encode the *combinations* of primitives, these neurons change their activity slowly. Fast context neurons change their activity quickly, encoding the primitives. Note that these different neural groups are related to the abstraction hierarchy of predictive coding (Section 2.4).

2.8.4.1 Benefit

Besides its ability to handle overlapping (sensory-motor) sequences with branching points, another benefit of the MTRNN is its ability to generalize between the learned sequences to a certain extent. Generalization is possible because the networks presented in [195], [230], [6] are actually dynamical systems. Changes in the initial activation state of context neurons lead to different trajectories in space-time, e.g. different sensory-motor sequences.

2.8.4.2 Limitation

A slight drawback of the MTRNN is that it needs preprocessing of input patterns and postprocessing of output patterns [230], [6]. Pre- and postprocessing consist of a sparse encoding of input patterns in order to reduce the pattern overlap and to consequently increase the learning ability of the network [230].
For robot control, the MTRNN cannot self-generate the actions. The actions must be shown to the system by a human teacher in a learning stage, for example through kinesthetic teaching of the robot.
Another disadvantage is the hyperparameterization of the MTRNN. The ratio of the number of neurons in the different neuronal groups and, more importantly, the ratio of the neuronal timescales strongly influence the learning performance of this network [230]. The determination of optimal network hyperparameters often involves a cumbersome trial-and-error process by a human experimenter.

2.8.5. Comparative Summary

I summarize the advantages and drawbacks of the different action selection mechanisms in Table 2.

2.9. Architectures for Sensory-Motor Learning

Based on the knowledge that I have presented in all previous sections of this chapter (Sections 2.1–2.8), I summarize and compare state-of-the-art architectures for sensory-motor learning. Some of the presented architectures may not be called *cognitive* architectures according to the strict definitions (Section 2.7.1) and the component requirements (Section 2.7.2). Nevertheless, all of these presented architectures facilitate sensory-motor learning on robots (simulated or real); that is what makes them highly relevant for the research presented in this thesis. The outcome of the comparison partly reflects the research gap that this thesis aims to fill.

Work	**Robot platform**	**Action selection mechanism**	**Benefit**	**Limitation**
Brooks [29]	Mobile robots (wheeled and insect-legged)	Implicit action selection: Hierarchical network of augmented finite state machines with suppressors and inhibitors	Robustness and fast reactions in dynamic environments	Scaling to more complex actions is difficult; learning is neglected since no memory
Maes [110]	Simulated agent	Implicit action selection: Activation/inhibition dynamics	Adaptivity to environmental and task features	Same as for Brooks [29]
Cheng et al. [42], [41]	Upper body ETL-humanoid	Integrative action selection: Dynamic integration of system components	Continuous, emergent behaviour	Learning is neglected since no memory
Vernon et al. [209], [211], [206], [207]	iCub humanoid	Motive-driven action selection: Explicit module for action selection	Increased autonomy for skill development	Action selection is de-coupled from action generation, this makes the adaptivity and flexibility difficult to evaluate
Yamashita and Tani [230]	Sony QRIO humanoid robot	Predictive coding based action selection: Changes in the activity of slow context neurons representing smooth transition between actions	Learning and recall of overlapping sensory-motor sequences with branching points; action selection is scalable	Relies on a human teaching every action; Estimation of the hyperparameters of MTRNN is difficult

Table 2 Comparative summary of different action selection mechanisms.

Note that this section is based on my description of related architectures for sensory-motor learning that I published in [222].

2.9.1. MOSAIC

Haruno et al. [70] propose the *MOSAIC* architecture. *MOSAIC* builds on the multiple paired forward and inverse models for motor control proposed by Wolpert and Kawato [227] (see Section 2.2 for forward and inverse models). In [70], *MOSAIC* is evaluated by simulations of an object manipulation task. However, development of related skills is neglected in this architecture.

2.9.2. HAMMER

Demiris and Dearden [53], and Demiris and Khadhouri [54] propose the *HAMMER* architecture. Like *MOSAIC*, the *HAMMER* architecture builds on the multiple paired forward and inverse models proposed in [227]. *HAMMER* uses prediction verification with confidence levels based on the prediction accuracy. Capabilities of *HAMMER* are the imitation of simple actions like gestures [53] and the recognition of manipulative actions, for example picking up an object [54].

2.9.3. Global Workspace

Shanahan [164] proposes the *Global Workspace* cognitive architecture. The architecture is based on the *Global Workspace* theory [11], [12], in which multiple processes corresponding to brain regions run in parallel and compete for access to a system-wide communication infrastructure referred to as *Global Workspace*. The theory provides a model for explaining consciousness and is substantiated by neuroscientific evidences. In addition, the theory proposes a solution to the frame problem[11] [165]. An action selection mechanism, corresponding to the basal ganglia of the brain, selects one “winner” process. The selection is guided by affect. One winner process can be active at a time. The winner process gets access to the global workspace. By doing so, the winner process closes the internal sensory-motor loop and passes data to all other competing processes, influencing the outcome of the competition. The competition of processes continues until another process is selected as winner, and so on.

The design of the *Global Workspace* architecture extends the combination of forward and inverse models by introducing two sensory-motor loops, an outer loop and an inner loop. The outer loop is closed physically by the environment and the inner (higher-order) loop is closed internally. The inner loop corresponds to the winner process of the *Global Workspace* theory. The inner loop realizes *mental simulation*[12] that is a key capability of the *Global Workspace* architecture. Mental simulation refers to the ability of going through several sensory-motor states in a closed-loop manner where the sensory effect of the motor output is inferred, without sending the motor output the robot’s actuators. Mental simulation realizes a type of pre-

[11]The frame problem comprises the quick extraction of relevant information or beliefs from a big set of data, along with the reasoning about the corresponding effects (or non-effects) of an agent’s action. See [165] for further references about the frame problem.

[12]In context of cognitive architectures, mental simulation is also referred to as *internal simulation*.

diction capability, where possible sensory states are inferred.
In [164], the architecture is implemented by generalized random access memories [2] and evaluated in a simulation environment. In the simulation, the architecture controls a simple mobile robot that can orient itself towards coloured cylinders depending on reward signals.

2.9.4. Predictive Coding-Based Architectures

The predictive coding approach (Section 2.4) supports the capability of mental simulation. Tani and his co-workers [190], [191], [126], [230], [6] realize mental simulation in neural networks based on predictive coding. These networks can be seen as *dynamic forward models* that predict visuo-proprioceptive sequences. Mental simulation is accomplished by feeding the (sensory-motor) output signals of the neurons back to their input. These predictive coding models show their strength by their capability of learning, recognizing, and imitating (manipulative) actions [6].

2.9.5. Staged Development Architecture — Bootstrapping from Learned Affordances

Ugur et al. [202] propose an architecture for staged development of robot skills. Given an *a priori* capability of hand-eye coordination and object detection at system start, a visually-guided robot arm can discover behaviour primitives. Key capability is the learning of object affordances. Based on the learned affordances, the architecture can bootstrap the imitation of manipulative actions.

2.9.6. Staged Development Architecture — Constraints Shaping Development

Based on [104], Law et al. [103] propose an architecture for a staged development that is shaped by constraint removal. Their architecture is designed for longitudinal development on the iCub robot. The main component of their architecture is the *Lift-Constraint, Act, Saturate (LCAS)* algorithm [102], [106] that is a contingency-based sensory-motor learner. The architecture starts with motor babbling and acquires hand-eye coordination and object manipulation skills.

2.9.7. Comparison of Architectures

For comparison purposes, I use criteria that are highly relevant for a cognitive architecture controlling a developmental robot. Many of these comparison criteria reflect the principle of ongoing emergence (Section 2.5) and the principle of verification and grounding (Section 2.6). I adopted comparison criteria for ongoing emergence from [143], and comparison criteria for verification and grounding from [180].
Table 3 and Table 4 added together show the comparison of state-of-the-art architectures for sensory-motor learning, both tables adopted from Wieser and Cheng [222].

Architecture	**Bootstrapping** of (initial) skills	**Continuous** skill acquisition	**Incorporation** of new skills	**Self-Verification** of signal or data	**Cognitive capability** by self-verification
MOSAIC [70]	✗	✓ (by linear interpolation among model outputs)	✗	prediction error, prior	selection of forward or inverse model
HAMMER [53], [54]	✓ (imitation)	✗	✗	prediction error, confidence	selection of forward or inverse model
Global Workspace [164]	✗	✗	✗	reward, saliency, veto	predictive action selection
Dynamic forward models [190], [191], [126], [230], [6]	✗	✗	✗	only in [6]: input-output states, context states	only in [6]: action recognition & planning
Dynamic integration [42], [41]	✗	✓ (by weighted integration of sensory-motor cues)	✗	✗	(*self-verification not addressed*)
Staged development: Bootstrapping from learned affordances [202]	✓ (imitation)	✓ (by staged acquisition)	✓ (by storing sensory-motor experiences)	prediction of action effects	action selection & planning
Staged development: Constraints shaping development [103]	✓ (hand-eye coordination)	✓ (by constraint removal)	✓ (by contextual memory)	history of sensory states	novelty detection
Proposed architecture in this thesis	✓ **(early hand-eye coordination)**	✓ **(by staged acquisition)**	✓ **(by training various new predictors)**	**prediction error, sensory-motor data, cumulative confidence representing predictor performance**	**action selection and mental simulation, adaptation to robot platform, self-collision prediction, reachability map acquisition**

Table 3 Comparison of architectures for robot skill acquisition, part 1. This table is adopted from Wieser and Cheng [222]. A checkmark (✓) indicates that a particular feature exists, a cross (✗) indicates that it does not exist.

Architecture	**Acquired robot skill**	**Improvement** in a particular skill	**Adaptation to robot** by adjusting proprioceptive feedback	**Robot platform(s)**
MOSAIC [70]	tracking a given trajectory by the arm while holding objects with unknown dynamics	✗	✗	none (skill evaluated in simulation)
HAMMER [53], [54]	imitation bootstrapped by babbling [53], recognition of demonstrated manipulative actions [54]	✗	✗	gripper [53], ActiveMedia Peoplebot [54] (has no manipulators)
Global Workspace [164]	robot turning to coloured objects depending on the predicted reward	✗	✗	simulated mobile robot
Dynamic forward models [190], [191], [126], [230], [6]	manipulative actions with generalization over object location [126], [230], [6], imitation of manipulative actions [6]	✗	✗	Sony QRIO [126], [230], HOAP-3 [6]
Dynamic integration [42], [41]	combining multiple pre-set skills, e.g. mimicking with arms while tracking by head	✗	✗	ETL-humanoid
Staged development: Bootstrapping from learned affordances [202]	discovery of behaviour primitives (*a priori* hand-eye coordination & object detection), affordance learning, imitation	✗	✗	7 DOF Motoman arm with 16 DOF Gifu hand, with tactile sensors and camera
Staged development: Constraints shaping development [103]	saccading & gazing, reaching, pointing, object interaction	✓ (smoother reaching)	✗	iCub
Proposed architecture in this thesis	**object tracking & evading by head, early reaching, improved reaching**	✓ **(robust blindfolded reaching, increased coverage of workspace)**	✓ **implications: increased reachability, self-collision avoidance**	**NAO [175], TOMM [52]**

Table 4 Comparison of architectures for robot skill acquisition, part 2. This table is adopted from Wieser and Cheng [222]. A checkmark (✓) indicates that a particular feature exists, a cross (✗) indicates that it does not exist.

2.10. Limitations of State-of-the-Art Systems and Open Issues

The summary of existing action selection methods and architectures for sensory-motor learning reveals some current limitations.
Older methods for action selection, for example [29], [110], support emergent actions but do not consider the aspect of development. Also, those methods have scalability issues when more complex behaviour should be supported.
A continuous generation of actions is supported by the integrative action selection [41], [42]. However, the method does not incorporate any newly generated action.
The action selection in biological agents is often guided by motives, pointed out early by Toda [199] and Pfeifer [137]. This concurs with the idea of autonomy (in the sense of self-preservation) of biological agents [113], [205]. Inspired by these findings, the action selection of the iCub cognitive architecture [211] supports a type of motive-driven selection. In the iCub architecture, the method generating the action itself (e.g. reach and grasp, locomotion) is considered separately from the method selecting the action. This de-coupling suggests that action selection and action generation are each realized by different methods. However, this may represent a drawback from the developmental point of view, since it would be difficult to accomplish a staged development of actions. It raises the question how can action selection develop if the generated actions themselves develop as well, and how do selection and generation mutually influence each other during development. Furthermore, the scalability of actions would be difficult, since the issue arises how a new class of action that is more complex or different than the previous ones can be incorporated into the architecture. This issue is substantiated by [211, p. 149], see "Guideline 42" about "hierarchically-structured representations for the acquisition, decomposition, and execution of action-sequence skills". Moreover, in [211], the actions realized by the framework of the iCub cognitive architecture are critically discussed:

> "A significant problem remains, however. In the existing framework, the actions that the robot uses to experiment with and explore the object are assumed to exist as predefined primitive manipulation movements, such as push, tap, and grasp. Clearly, we require a more flexible approach in which the action's movements can be generated (...)." [211, p. 156]

In sum, the action selection methods presented in [29], [110], [42] neglect the development of actions. For action development, methods of self-exploration generating training samples (e.g. [153], [18]) provide an alternative solution but do not consider robustness to the temporary loss of sensory features, for example visual occlusion.
The predictive coding-based action selection [230] is promising, since it encompasses selection (in terms of switching between actions) and generation (in terms of action primitives) in one coherent technical framework (artificial neural network with temporal scales). Its advantage is the scalability. Since predictive coding-based neural networks share the same or similar internal representation, they can be interfaced together easier than the distinct

modules of architectures like the iCub cognitive architecture in [211]. However, one slight limitation is the estimation of the hyperparameters of networks with spatial and temporal scales. So far, the estimation of hyperparameters involved a human expert who conducts a series of trial-and-error experiments until the optimal parameters are found. Another limitation of the predictive coding methods by Tani [192] is that they require a human teacher for the acquisition of any action or skill. While this is an obvious necessity for higher-level procedural skills, for example the stacking of toy bricks, it remains a limitation for lower-level *coordination* skills like object tracking by head/eyes or reaching to moving targets, in other words, for hand-eye coordination. Humans acquire hand-eye coordination skills *autonomously* without a caregiver. The predictive coding methods summarized by [192] neglect the learning of *coordination* skill, since it was given by a pre-programmed vision tracking system, for example to track an object by head, as in [230], [126], [195].

An important ingredient for the acquisition of hand-eye coordination skills is the learning of dynamics, both a dynamics of the self and a dynamics of external entities such as moving objects. In the predictive coding approach by Friston [59], [60], the hidden states link dynamics over time. In [60, p. 135], the dynamics is already provided, see the equations of motion of the “mountain car problem”. However, a developmental agent should be able to learn such necessary dynamics by experience, for example by observing possible motions during interaction.

The action selection method is a crucial part of a developmental cognitive architecture. Only few architectures, like [103], [202], support a staged development where the generated data or skills of one particular stage serve as prior experience for the next stage. Tables 3 and 4 reveal that none of the architectures except for one (by Law et al. [103]) support an improvement of an already existing skill. It should be noted that development is a complex two-fold process, two-fold in the sense that new skills arise on the one hand, and already existing skills improve on the other hand. Furthermore, none of the presented architectures facilitate an adaptation to the robot’s morphology. This is an important aspect, since the transfer of an architecture to another robot platform involves the tuning of a set of parameters by a human expert, in many cases parameters that regulate the control of the joints that vary from robot to robot.

Finally, Table 5 summarizes the limitations of existing systems with regard to action selection and architectures, both from a developmental point of view, and to what extent these limitations are overcome by my proposed system in this thesis.

Related work	**Category**	**Limitation**	**Limitation overcome by proposed work**
Brooks [29], Maes [110]	action selection level	Does not develop new behaviours or skills	Bootstraps basic sensory-motor skills
Friston [60, p. 135]	action selection level	Dynamics given beforehand (these should be rather learned by the agent through observation)	Trajectories of moving objects can be learned through observation
Works on learning by self-exploration, e.g. intrinsic motivation by Oudeyer and co-workers [132], [17], [18], goal babbling by Rolf and co-workers [148], [147], motor babbling by Saegusa and co-workers [153], [151]	action selection level, architecture level	Loss of sensory data (e.g. visual occlusion) is not considered	Robustness to loss of sensory (visual) data: sensory-motor experience yields robustness to long-term loss of camera vision and occlusion/covering of end-effector; learning the observed trajectories of moving objects yields robustness to short-term occlusion/covering of external object
Vernon and co-workers [209], [211], [206], [207]	action selection level, architecture level	Bootstrapping/generation as well as incorporation of action and action sequences are missing	Bootstraps basic sensory-motor skills; newly generated sensory-motor sequences can be learned/ incorporated
Tani and co-workers [190], [191], [126], [230], [6]	action selection level, architecture level	No support of an *autonomous* acquisition of *coordination* skills (e.g. object tracking by head, reaching to *moving* targets); manual estimation of network *hyperparameters* is difficult (e.g. MTRNN [230])	Acquires coordination skills autonomously (bootstrapping) and improves them (e.g. faster reaching, blindfolded reaching); supports an autonomous estimation of network hyperparameters
Staged development, e.g. Ugur et al. [202], Law et al. [103]	architecture level	Some basic skills given beforehand (in [202]) instead of generating them; embodiment: the adaptation to different robot platforms is not addressed; robustness is not addressed, e.g. robust behaviour despite temporal loss of sensory data	Adjustment of proprioceptive feedback and scalability to available number of DOF together yield adaptation to different robot platforms

Table 5 Limitations of existing systems (related work) with regard to action selection and architectures for sensory-motor learning. Note that my proposed work is settled in both action selection level and architecture level.

3. Chapter

Evolutionary Optimized Multiple Timescale Recurrent Neural Network

The benefits of predictive coding (Section 2.4) substantiate the necessity of a method for spatiotemporal learning in my proposed cognitive architecture. On the computational level, RNNs offer one way to model predictive coding (Section 2.4.3). However, the limitations in Section 2.8.4 and Section 2.10 point out that an important type of RNN, the MTRNN, requires a good guess of hyperparameters to function well.
In this chapter, I propose a solution to this problem of hyperparameter estimation for predictive coding-based RNNs. My solution is based on the MTRNN. The outcome is an extended version of the MTRNN, referred to as evolutionary optimized MTRNN (EO-MTRNN) that can automatically estimate its hyperparameters. Besides its ability of autonomous hyperparameter estimation, the EO-MTRNN adds the option for early stopping to its learning process, and does not require additional neural networks for pre- and postprocessing the input-output data. The EO-MTRNN is a key building block of my proposed new action selection system that is explained later in this thesis.
This chapter has the following structure: Section 3.1 highlights the significance of the MTRNN for spatiotemporal learning. Section 3.2 summarizes its existing limitation that is mainly the handling of the network in terms of hyperparameter estimation. Section 3.3 presents my approach to tackle the limitation. The approach is characterized by the idea of a self-improving spatiotemporal learner. Section 3.4 briefly reviews potential methods that can be used to solve the problem of hyperparameter estimation and explains my decision for a particular optimization method. Section 3.5 and Section 3.6 together contain a detailed system description of my proposed EO-MTRNN. Regarding the performance evaluation, Section 3.7 describes a benchmark dataset that I proposed for an empirical analysis of the network. The experiments encompass a validation of the proposed network by using the proposed benchmark dataset as well as sensory-motor data from a real robot experiment. Section 3.8 shows the obtained results. Section 3.9 provides a thorough discussion of the obtained results. Finally, Section 3.10 summarizes of this chapter.
Note that I published parts of this chapter in [223].

3.1. The Significance of the MTRNN for Spatiotemporal Learning

In general, some machine learning and robotics applications require the learning of sequences. Here, a sequence consists of (spatial) patterns that change over time. Such

sequence is also referred to as spatiotemporal pattern. Mathematically, a spatiotemporal pattern can be expressed by a time series of vectors. The dimension of each vector is the spatial dimension and stays constant over time. The length of the sequence, i.e. the number of vectors in the series, is the temporal dimension. A formal notation of a spatiotemporal pattern S is: $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{L-1}$ with $\mathbf{x}_i^T = (x_0 \quad x_1 \quad x_2 \quad ... \quad x_{N-1})$, with L denoting the temporal dimension and N denoting the spatial dimension.
Spatiotemporal learning is supported by dynamical systems, in particular RNNs. Important types of RNNs encompass Elman networks [57], Jordan networks [79], [80], echo state networks [75], [76], and long short-term memory (LSTM) [72] that overcomes the vanishing gradient problem [65].
Nevertheless, when applying a RNN to control a robot, it is desirable to achieve more than only sequence learning. Some kind of goal-directedness should be represented in the network in order to support goal-directed actions [126]. Also, the switching or transition between different actions should happen smoothly and be encoded in the network. This implies a functional hierarchy: actions in the lower-level and transitions between them in the higher-level.
In order to study the emergence of functional hierarchy for action learning and generation in robotics, Yamashita and Tani [230] propose the MTRNN model. The MTRNN fits into the predictive coding framework [60], [192], since it can learn, generate, and recognize spatiotemporal patterns by prediction error minimization. Recognition means that (optimal) initial states of the context neurons can be found, leading to the (re)generation of patterns that match the observed ones. This recognition capability is achieved by an iterative search procedure [6]. Moreover, the principle of predictive coding emphasized that the top-down predictions and the bottom-up errors propagate in a hierarchical structure. That hierarchical structure is represented by the functional hierarchy of the MTRNN.
The application possibilities offered by the MTRNN as a dynamic system and the idea of multiple timescales are investigated in [6], [231], [77]. Further MTRNN applications are shown in [155], [189], [188].
Note that action learning is only a particular application of the MTRNN. In general, the network facilitates the hierarchically-structured learning of sequences, i.e. the lower-level representations do not necessarily have to encode actions; they can encode any abstract or non-abstract entities (such as characters or words [71]) that should be arranged in a sequence.
In detail, the functional hierarchy of the MTRNN can be expressed by two characteristics: spatiotemporal *compositionality* and *self-organization of higher-level representations*.

3.1.1. Compositionality

Compositionality refers to the combination of entities in order to form sequences. Applied to action selection (Section 2.8.4), the entities are action primitives. Selection is achieved by a smooth transition between these primitives. A traditional approach uses many separate local networks encoding the action primitives and a higher-level selection module to switch between the them, for example in [196]. In [196], the local networks as well as the higher-level selection module are each implemented by a CTRNN. Compared to such an approach, the benefit of the MTRNN is that it integrates both capabilities, the encoding of primitives and the smooth transition between them, into one network model. This is achieved through its property of multiple timescales [230]: Changes in sensory-motor trajectories that occur over short time ranges require a low value of the timescale τ. In order to preserve higher-order information, e.g. action goals over longer trajectories with branching, a larger value of the timescale τ is required. Thus, the MTRNN overcomes the limited storage capacity of a standard CTRNN that operates with only one group of context neurons with the same timescale.

In sum, the MTRNN achieves compositionality by encoding primitive sequences in a group of context neurons with a faster change in neuronal activity (fast context) compared to another group of context neurons that has a slower change in activity (slow context). The slow context neurons alter their activity when a transition between primitive sequences occurs, which is for example the case at branching points of trajectories in space-time.

3.1.2. Self-Organization of Higher-Level Representations

Predictive coding contains hierarchical representations. In the cortex, higher-levels of the cortical hierarchy contain abstract entities such as goals or intentions [62], [16]. These higher-levels contain representations that are updated through the bottom-up error propagation process [60].

When learning spatiotemporal sequences, the MTRNN creates an internal representation of functional hierarchy. The functional hierarchy is accomplished by the self-organization of the initial activation states of the context neurons. Note that for a functional hierarchy to emerge, at least two groups of context neurons have to be present in the network. One group operates with a particular timescale that is different from the timescale of another group. In case of two context groups, like in the original MTRNN, one context group has a higher value of its timescale than the other context group. The initial activation state of each context neuron self-organizes itself as a result of the error propagation process. Mathematically, the update procedure of the initial activation state of each single context neuron contains two steps: first, an update of the potential state of the neuron, and second, the update of the activation function of the neurons given the potential state as input.

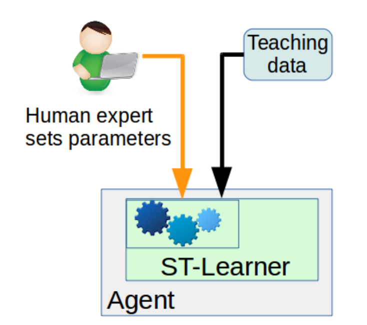
3.2. Limitation of the MTRNN

The MTRNN model is based on the aforementioned CTRNN. In fact, the original MTRNN proposed in [230] is a special type of CTRNN that operates with two different timescales for the context neurons, yielding two different context groups, fast context and slow context. The MTRNN uses sigmoid activation in the fast and slow context group, and softmax activation in the input-output group. The MTRNN input and output data are processed by topology preserving maps that are SOMs [88], [89]. This pre- and postprocessing of input and output data by topology preserving maps has always been necessary when using the MTRNN, for example in [230], [6]. According to [230], the pre- and postprocessing of the network input and output increases the learning ability by reducing the overlaps in the training data.

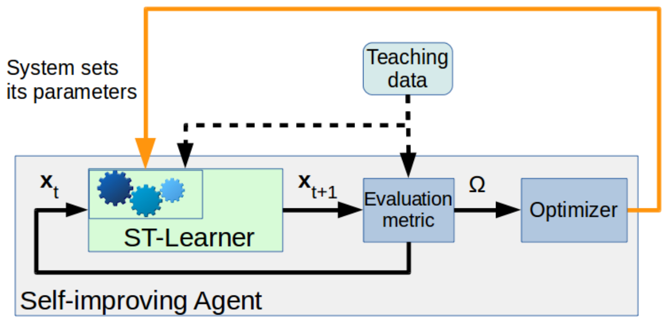
However, the usage of topology preserving maps represents a slight drawback of the original MTRNN, since the topology preserving maps are neural networks that need to be trained as well in addition to the main network. It has not been investigated how another version of the MTRNN performs that does not require training of additional networks for pre- and postprocessing.
The learning performance of the MTRNN is strongly dependent on the settings of the timescales for each neuron group; in particular, the ratio between the timescales for fast context and slow context influences the performance [230]. Networks related to the MTRNN, such as the network proposed in [82], also rely on multiple timescales and it is crucial to find good values for them. However, finding proper values for these different timescales is problematic and so far it has been achieved through many trial-and-error tests conducted by a human experimenter who evaluated the network performance.

3.3. Proposed Approach: Self-Improving Spatiotemporal Learner

Suppose a spatiotemporal learning method (abbreviated by ST-learner) is a crucial part of an agent (e.g. robot) interacting with its surroundings. For example, the ST-learner may learn from collected sensory-motor samples (teaching data) how to control the agent in a meaningful way and to make predictions in various sensory-motor contexts, these capabilities are later helpful for skill sharing and scaffolding [216]. The ST-learner should form internal representations and, more importantly, be able to relearn over the agent's lifetime. When the teaching data are changing or are updated, the learner should relearn to incorporate the changes. If data sets change rapidly in both spatial and temporal dimension, then learner will have to be reparameterized to capture the new data. However, this reparameterization is done by a human programmer who understands the task and has the necessary experience of deploying the learner. In contrast, a developmental agent, situated in the world through sensors and motors, is required to manage the parameterization on its own, without the necessity to manually tune its learning mechanism by a human expert [213]. This line of thinking concurs with Vernon et al. [210] who conclude that developmental agents must be capable of self-modification, both in terms of parameter adjustment (i.e. learning data given the system structure) and in terms of hyperparameter adjustment (i.e. modify the parameters that form the system structure). The adjustment of hyperparameters means altering the system dynamics, thus influencing the desired performance.
Consider the MTRNN as a spatiotemporal learning module that is part of a higher-level learning system, for example a system controlling an agent. The autonomous hyperparameter estimation would enable the ST-learner to reconfigure itself during environmental interaction, fitting the latest set of data collected for learning.
Figure 4 shows my proposed concept that is applied to the MTRNN as ST-learner.
Each concept, traditional and proposed, has a different implication when the system is put into action, see Figure 5.

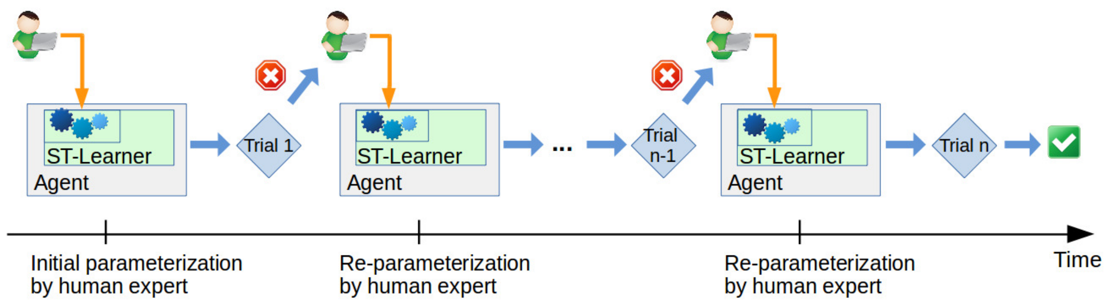


(a) Traditional concept of a ST-learner

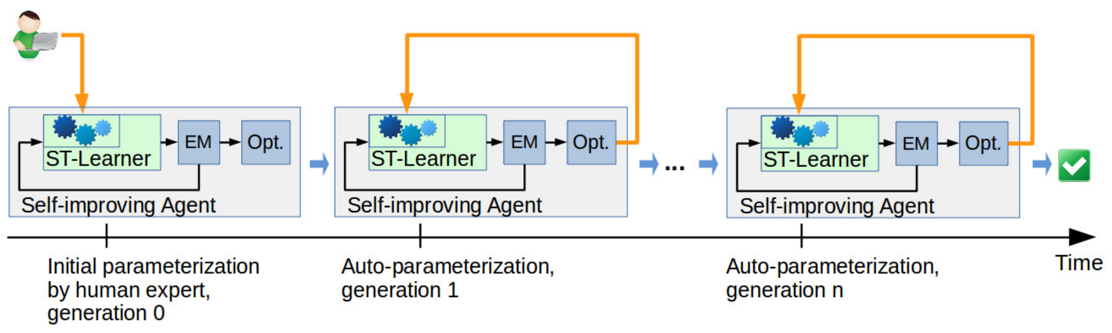


(b) Proposed concept of a ST-learner

Figure 4 Concepts of a spatiotemporal learner (ST-learner) inside an agent. Traditional applications (4(a)) require a human expert to parameterize the learning system. My proposed concept (4(b)) extends a ST-learner by AHE, yielding a self-improving agent. Note that the ST-learner can be a neural network.



(a) Scenario of the traditional ST-learner concept. Human expert required in the loop until system shows satisfactory performance.



(b) Scenario of the proposed ST-learner concept. Human expert only required to initialize the system.

Figure 5 Implications of the different spatiotemporal learner (ST-learner) concepts (to avoid clutter, the "Teaching data" box is omitted). Scenario 5(a) means cumbersome trial-and-error procedures involving a human expert until the learner performs well enough. In contrast, my proposed concept of AHE leads to scenario 5(b), in which the system improves itself over several generations through time.

3.4. Optimization Methods

Self-improvement over multiple generations through time is considered as an optimization problem that can be approached by state-of-the-art optimization methods.
I consider optimization methods that neither depend on an analytical description of the optimization problem nor on its gradient. Evolutionary algorithms and evolutionary optimization become increasingly popular, since they do not require information about the fitness landscape, work in high-dimensional search spaces, and can be parallelized [233].
A common optimization method is random search [4], [176], [22], although it is not necessarily considered as evolutionary. Greff et al. [68] use random search to optimize the hyperparameters of LSTM networks.
Another optimization method for very large search spaces is particle swarm optimization (PSO) [86], [56], [167]. However, it is not guaranteed that PSO finds an optimal solution, often the user has to make a tradeoff between exploration and exploitation [201].
A promising candidate for global optimization is differential evolution (DE) [178], [177], [179]. Similar to PSO, DE can be applied to a wide variety of numerical optimization problems with very high dimensions, noise, and fluctuations over time, such as for example in [146]. In [212], [28], it is reported that DE outperforms PSO in the quality of computed solutions on benchmark problems. However, as it is the case with other optimization methods, the performance of DE is dependent on its control parameters. Adverse values for these parameters deteriorate the optimization performance.
To overcome this drawback, Liu and Lampinen [107] propose a fuzzy adaptive version of DE, the fuzzy adaptive differential evolution (FA-DE), which outperforms the original DE.
Brest et al. [28] propose a version of DE with self-adapting control parameters, referred to as self-adaptive differential evolution (SA-DE). They have extensively compared the performance of SA-DE with FA-DE and other related methods. They report that SA-DE yielded better results than FA-DE, and that SA-DE yielded better or at least comparable results than the other evolutionary algorithms proposed in [232], [105].
These findings suggest that SA-DE is the ideal candidate for the optimization of MTRNN hyperparameters.

3.5. EO-MTRNN — Part 1: The Modified MTRNN

In order to realize an easier handling of the MTRNN, I propose a modified version that does not require training of additional neural networks (such as topology preserving maps) for pre- and postprocessing.
This section describes the proposed version by showing the network structure (Section 3.5.1) and explaining the main modification of the original version. For the modified version, I derive a detailed mathematical description of its training algorithm (Section 3.5.2). Then, I propose an early stopping method (Section 3.5.3) that can be optionally included into the training process. Furthermore, I suggest an analytical method for preprocessing the input and post-

processing the output data (Section 3.5.4).

In the following, I use the following acronyms unless defined otherwise:
IO: input-output group
C: context group, can be split into fast context (FC) and slow context (SC)
$N_{IO} \in \mathbb{N}$: number of input-output units
$N_{FC} \in \mathbb{N}$: number of fast context units
$N_{SC} \in \mathbb{N}$: number of slow context units
$N_S \in \mathbb{N}$: number of sequences
$L \in \mathbb{N}$: sequence length
$E \in \mathbb{R}$: loss function
$\hat{y} \in \mathbb{R} \mid 0.0 < \hat{y} < 1.0$: sample value (of a spatial dimension) of a training sequence
$y \in \mathbb{R} \mid 0.0 < y < 1.0$: activation value of an input-output unit
$u \in \mathbb{R}$: potential value of an input-output unit
$x \in \mathbb{R} \mid 0.0 < x < 1.0$: input value fed into an input-output unit
$c \in \mathbb{R} \mid 0.0 < c < 1.0$: activation value of a context unit
$q \in \mathbb{R}$: potential value of a context unit
$\tau \in \mathbb{N}_{>0}$: timescale of a unit
$w_{ab}^{(ik)} \in \mathbb{R}$: connective weight from unit $b^{(k)}$ to unit $a^{(i)}$, with i, k as indexes
$t \in \mathbb{N}$: time step

3.5.1. Network Structure

Figure 6 shows the structure of the proposed modified version of the MTRNN. The cardinality of set F_1 is denoted as $|F_1|$. It is $|F_1| = N_{IO} + N_{FC} + N_{SC}$. The set F_2 has the same cardinality than F_1.
Compared to the original version by Yamashita and Tani [230] using softmax activation for the input-output group, one difference here is the usage of sigmoid activation for the input-output group of this network. The activation of the input-output group is now consistent with the activation of the context group that has already had sigmoid activation in the original version. The entire network now consists of units with sigmoid activation, making it uniform. Besides uniformity, I propose to use the sigmoid activation for the input-output group because of the universal approximation theory of neural networks. Any continuous function can be approximated by sigmoid units [51], [19]. For a dynamic network, such as the MTRNN, this implies controllability, i.e. any desired state can be achieved within a finite number of steps starting from an initial state. The original version uses softmax activation for the input-output group, since, as described in [230], the softmax activation fits well to the pre- and postprocessor networks (topology preserving maps [88], [89]) connected to the MTRNN. However, these pre- and postprocessor networks have to be trained as well, in addition to the main network. In contrast, this proposed version does not require pre- and postprocessor networks.
Nevertheless, if pre- and postprocessing is requested to introduce a sparse representation of the actual input data, it can be realized by a simple analytical pre- and postprocessing scheme. I propose this analytical preprocessing scheme as one of the network configuration

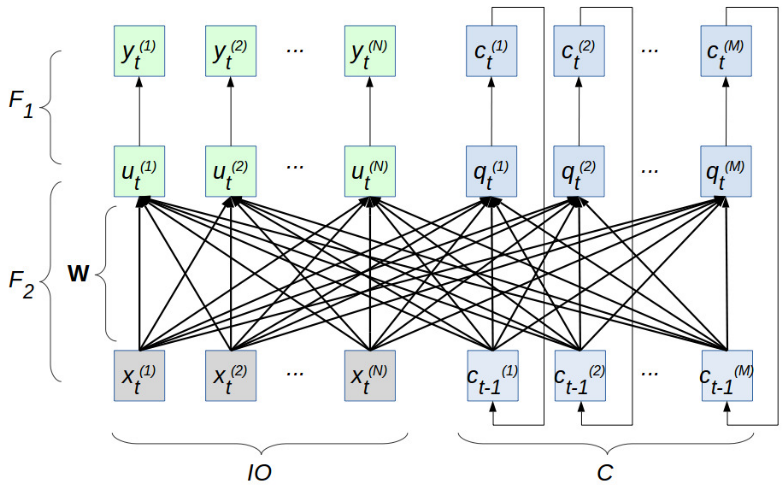


Figure 6 Proposed version of the MTRNN working with sigmoid activation for all units. The network consists of units connected by a set of functions F_1 and F_2. The left half shows the input-output group IO and the right half shows the context group C. The set F_1 represents the activation functions which are $y(u)$ for the IO group and $c(q)$ for the C group. The set F_2 represents the functions for updating the potentials which are $u_t(\tau, u_{t-1}, w, x, c_{t-1})$ for the IO group and $q_t(\tau, q_{t-1}, w, x, c_{t-1})$ for the C group. The F_2 functions include recurrent dependencies, weights, and timescales. The thick arrows between the bottom and the middle units indicate the connective weights represented by the weight matrix $\mathbf{W}$. The feedback connections from the top to the bottom context units indicate the recurrent connections of the output of context neurons. Note that the timescales can be different from each other, splitting the C group into FC and SC. Also note that the connections between the IO group and the SC group are set to zero. The extension of this network by an evolutionary optimization method yields the enhanced version (EO-MTRNN) which is shown in Figure 7.

modes. The second proposed mode is an early stopping method included into the training procedure. When activated, it is supposed to reduce the overfitting of the teaching data.

3.5.2. Training Algorithm

I use sigmoid neurons throughout the entire network.

The activation of an input-output neuron is given by Equation (1) and the activation of a context neuron is given by Equation (2).

$$y(u) = \frac{1}{1+\exp(-u)} \tag{1}$$

$$c(q) = \frac{1}{1+\exp(-q)} \tag{2}$$

The potential value of an input-output unit is updated by

$$u_t^{(i)} = \Big(1 - \frac{1}{\tau^{(i)}}\Big) u_{t-1}^{(i)} + \frac{1}{\tau^{(i)}} \Big(\sum_{k \in IO} w_{ux}^{(ik)} x_t^{(k)} + \sum_{k \in C} w_{uc}^{(ik)} c_{t-1}^{(k)} \Big) \tag{3}$$

with $i \in IO$.

The potential value of a context unit is updated by

$$q_t^{(i)} = \Big(1 - \frac{1}{\tau^{(i)}}\Big) q_{t-1}^{(i)} + \frac{1}{\tau^{(i)}} \Big(\sum_{k \in IO} w_{qx}^{(ik)} x_t^{(k)} + \sum_{k \in C} w_{qc}^{(ik)} c_{t-1}^{(k)} \Big) \tag{4}$$

with $i \in C$.

I use the following loss function:

$$E = \sum_t E_t = \sum_t \sum_{i \in IO} \frac{1}{2}\Big(\hat{y}_t^{(i)} - y_t^{(i)}\Big)^2 \tag{5}$$

In the following, I describe all partial derivatives that are required for the update of neural weights and the update of initial potentials of context neurons.
The description is based on the network structure (Figure 6), the functions (1), (2), (3), (4), and the loss function (5).
The partial derivative $\frac{\partial E}{\partial u_t^{(i)}}$ can be expressed as:

$$\frac{\partial E}{\partial u_t^{(i)}} = \frac{\partial E_t}{\partial u_t^{(i)}} + \frac{\partial}{\partial u_t^{(i)}}\Bigg(\sum_{t'=t+1} E_{t'}\Bigg) \tag{6}$$

Equation (6) is the recurrence equation of the input-output group and it is developed to contain the recurrence term $\frac{\partial E}{\partial u_{t+1}^{(i)}}\frac{\partial u_{t+1}^{(i)}}{\partial u_t^{(i)}}$. It follows

$$\frac{\partial E}{\partial u_t^{(i)}} = \frac{\partial E_t}{\partial u_t^{(i)}} + \frac{\partial E}{\partial u_{t+1}^{(i)}}\frac{\partial u_{t+1}^{(i)}}{\partial u_t^{(i)}} \tag{7}$$

with $i \in IO$.
The right side of Equation (7) is going to be expanded.
Applying the chain rule to $\frac{\partial E_t}{\partial u_t^{(i)}}$ and deriving $\frac{\partial u_{t+1}^{(i)}}{\partial u_t^{(i)}}$ by using Equation (3) result in:

$$\frac{\partial E}{\partial u_t^{(i)}} = \frac{\partial E_t}{\partial y_t^{(i)}}\frac{\partial y_t^{(i)}}{\partial u_t^{(i)}} + \frac{\partial E}{\partial u_{t+1}^{(i)}}\Big(1 - \frac{1}{\tau^{(i)}}\Big) \tag{8}$$

It follows for one unit $i \in IO$:

$$\frac{\partial E}{\partial y_t^{(i)}} = \frac{\partial}{\partial y_t^{(i)}}\bigg(\frac{1}{2}\Big(\hat{y}_t^{(i)} - y_t^{(i)}\Big)^2\bigg) = -\Big(\hat{y}_t^{(i)} - y_t^{(i)}\Big) \tag{9}$$

The variable $y_t^{(i)}$ denotes the sigmoid activation of the IO group (Equation (1)), thus one can write:

$$\frac{\partial y_t^{(i)}}{\partial u_t^{(i)}} = \frac{\partial}{\partial u_t^{(i)}}\left(\frac{1}{1+\exp(-u_t^{(i)})}\right) = y_t^{(i)}\Big(1 - y_t^{(i)}\Big) \tag{10}$$

Inserting Equation (9) and Equation (10) back into Equation (8) yields

$$\frac{\partial E}{\partial u_t^{(i)}} = \Big(y_t^{(i)} - \hat{y}_t^{(i)}\Big)y_t^{(i)}\Big(1 - y_t^{(i)}\Big) + \Big(1 - \frac{1}{\tau^{(i)}}\Big)\frac{\partial E}{\partial u_{t+1}^{(i)}} \tag{11}$$

with $i \in IO$.

In analogy to Equation (6), the partial derivative $\frac{\partial E}{\partial q_t^{(i)}}$ can be expressed as

$$\frac{\partial E}{\partial q_t^{(i)}} = \sum_{k \in IO} \frac{\partial E}{\partial u_{t+1}^{(k)}} \frac{\partial u_{t+1}^{(k)}}{\partial c_t^{(i)}} \frac{\partial c_t^{(i)}}{\partial q_t^{(i)}} + \sum_{k \in C} \frac{\partial E}{\partial q_{t+1}^{(k)}} \frac{\partial q_{t+1}^{(k)}}{\partial q_t^{(i)}} \tag{12}$$

with $i \in C$.

Equation (12) is the recurrence equation of the context group, resulting from the structure and connectivity of the network. The right side of Equation (12) contains the following parts:

$$\frac{\partial u_{t+1}^{(k)}}{\partial c_t^{(i)}} = \frac{1}{\tau^{(k)}} w_{uc}^{(ki)} \tag{13}$$

with $i \in C$ and $k \in IO$,

$$\frac{\partial c_t^{(i)}}{\partial q_t^{(i)}} = \frac{\partial}{\partial q_t^{(i)}} \left(\frac{1}{1 + \exp(-q_t^{(i)})} \right) = c_t^{(i)} \left(1 - c_t^{(i)} \right) \tag{14}$$

with $i \in C$,

$$\frac{\partial q_{t+1}^{(k)}}{\partial q_t^{(i)}} = \delta_{ik} \left(1 - \frac{1}{\tau^{(k)}} \right) + \frac{1}{\tau^{(k)}} w_{qc}^{(ki)} \frac{\partial c_t^{(i)}}{\partial q_t^{(i)}} \tag{15}$$

with $i \in C$, $k \in C$, and δ_{ik} as Kronecker delta ($\delta_{ik} = 1$ for $i = k$, $\delta_{ik} = 0$ for $i \neq k$).

Inserting Equation (14) into Equation (15) results in

$$\frac{\partial q_{t+1}^{(k)}}{\partial q_t^{(i)}} = \delta_{ik} \left(1 - \frac{1}{\tau^{(k)}} \right) + \frac{1}{\tau^{(k)}} w_{qc}^{(ki)} c_t^{(i)} \left(1 - c_t^{(i)} \right) \tag{16}$$

with $i \in C$ and $k \in C$.

Inserting Equation (13), Equation (14), and Equation (16) back into Equation (12) yields

$$\begin{aligned} \frac{\partial E}{\partial q_t^{(i)}} = & \sum_{k \in IO} \frac{\partial E}{\partial u_{t+1}^{(k)}} \frac{1}{\tau^{(k)}} w_{uc}^{(ki)} c_t^{(i)} \left(1 - c_t^{(i)} \right) + \\ & \sum_{k \in C} \frac{\partial E}{\partial q_{t+1}^{(k)}} \left(\delta_{ik} \left(1 - \frac{1}{\tau^{(k)}} \right) + \frac{1}{\tau^{(k)}} w_{qc}^{(ki)} c_t^{(i)} \left(1 - c_t^{(i)} \right) \right) \end{aligned} \tag{17}$$

with $i \in C$.

The partial derivatives $\frac{\partial E}{\partial u_t^{(i)}}$ (Equation (11)) and $\frac{\partial E}{\partial q_t^{(i)}}$ (Equation (17)) are important, since they are required to compute the gradients $\frac{\partial E}{\partial w}$ according to:

$$\frac{\partial E}{\partial w_{ux}^{(ik)}} = \sum_t \frac{\partial E}{\partial u_t^{(i)}} \frac{\partial u_t^{(i)}}{\partial w_{ux}^{(ik)}} = \sum_t \frac{\partial E}{\partial u_t^{(i)}} \frac{1}{\tau^{(i)}} x_t^{(k)} \tag{18}$$

with $i, k \in IO$,

$$\frac{\partial E}{\partial w_{uc}^{(ik)}} = \sum_t \frac{\partial E}{\partial u_t^{(i)}} \frac{\partial u_t^{(i)}}{\partial w_{uc}^{(ik)}} = \sum_t \frac{\partial E}{\partial u_t^{(i)}} \frac{1}{\tau^{(i)}} c_{t-1}^{(k)} \tag{19}$$

with $i \in IO, k \in C$,

$$\frac{\partial E}{\partial w_{qx}^{(ik)}} = \sum_t \frac{\partial E}{\partial q_t^{(i)}} \frac{\partial q_t^{(i)}}{\partial w_{qx}^{(ik)}} = \sum_t \frac{\partial E}{\partial q_t^{(i)}} \frac{1}{\tau^{(i)}} x_t^{(k)} \tag{20}$$

with $i \in C, k \in IO$,

$$\frac{\partial E}{\partial w_{qc}^{(ik)}} = \sum_t \frac{\partial E}{\partial q_t^{(i)}} \frac{\partial q_t^{(i)}}{\partial w_{qc}^{(ik)}} = \sum_t \frac{\partial E}{\partial q_t^{(i)}} \frac{1}{\tau^{(i)}} c_{t-1}^{(k)} \tag{21}$$

with $i \in C, k \in C$.

Given these partial derivatives, the weights and initial potentials can be updated as part of the BPTT algorithm [150], [149]. Within BPTT, the partial derivatives (11), (17) are iteratively computed for each training sequence and are required to compute the partial derivatives of the weights. The partial derivatives (18), (19) (20), (21) are iteratively computed for each training sequence with t as sample index and are then summed up over the training sequences s given in the training set S.

At the beginning of training, the procedure initializes the weights with random values between -0.025 and 0.025 like in [230]. In the multiple timescale mode, i.e. $\tau_{FC} \neq \tau_{SC}$, the connective weights between the IO group and the SC group are set to zero. The weights are updated by

$$\begin{aligned} \Delta w_{n+1} &= \alpha \frac{1}{T} \sum_{s \in S} \frac{\partial E^{(s)}}{\partial w} + \eta \Delta w_n \\ w_{n+1} &= w_n - \Delta w_{n+1} \end{aligned} \tag{22}$$

where n is the epoch index, T is the number of total samples of the training set, α is the learning rate of weight update, and η is the momentum. When the BPTT procedure arrives at the very first sample of a sequence, an additional backpropagation step is required in order to compute the initial potentials of the context neurons by

$$\begin{aligned} \frac{\partial E}{\partial q^{\star(i)}} = &\sum_{k \in IO} \frac{\partial E}{\partial u_0^{(k)}} \frac{1}{\tau^{(k)}} w_{uc}^{(ki)} c^{\star(i)} \left(1 - c^{\star(i)}\right) + \\ &\sum_{k \in C} \frac{\partial E}{\partial q_0^{(k)}} \left(\delta_{ik} \left(1 - \frac{1}{\tau^{(k)}}\right) + \frac{1}{\tau^{(k)}} w_{qc}^{(ki)} c^{\star(i)} \left(1 - c^{\star(i)}\right)\right) \end{aligned} \tag{23}$$

with $i \in C$ and δ_{ik} as Kronecker delta. In Equation (23), $c^{\star(i)}$ represents the initial input activation value of a context neuron i. The value of $c^{\star(i)}$ is set to 0.5 before the training starts. An activation value of 0.5 corresponds to a potential of 0.0 for a sigmoid neuron. It is considered as a neutral value from which the initial potential is adapted through the training process.

For the set S of given training sequences, the initial context potentials $q^\star$ are updated by

$$q_{n+1}^{\star(si)} = q_n^{\star(si)} - \beta^{(i)} \frac{1}{L_s} \frac{\partial E^{(s)}}{\partial q^{\star(i)}} \tag{24}$$

where $s \in S$, $i \in C$, n is the epoch index, L_s is the sequence length, and $\beta^{(i)}$ is the learning rate of potential update. Note that $\beta^{(i)}$ can have two different values, depending on whether the unit i is part of the fast context or slow context group.
The training procedure also contains a slight bias by clipping the gradients (11), (17), (23) through $\delta_c := tanh(\delta)$ where in this case δ represents the gradient (11), (17), (23), respectively. The weight gradients (18), (19), (20), (21) as well as the initial potentials (24) are then computed using the corresponding clipped gradient δ_c. According to [67, p. 409], the usage of clipped gradients stabilizes the training procedure by preventing an accumulation of too high values for the partial derivatives.
I use the mean squared error (MSE) as part of the BPTT training procedure. The reason for choosing the MSE resides in machine learning theory. The basis for supervised learning is the maximum likelihood estimator that aims to maximize a log-likelihood. Maximizing the log-likelihood with respect to the model parameters yields the same estimate of parameters than minimizing the MSE, as explained in [67, p. 132].
In the recall phase after training, when an input sample is fed into the input-output group of the network, the entire context states (fast and slow context) are recognized through an iterative value search and initialized. Given a particular sequence S_g and an input pattern $\mathbf{x}_t$, the context recognition starts with setting the initial IO and the initial C activation states from the BPTT training procedure. By closed-loop prediction, where the network output is fed into the input, subsequent IO patterns are computed, each with its corresponding C activation. The predicted IO pattern that has the minimum Euclidean distance to the given input pattern $\mathbf{x}_t$ is selected and its corresponding C activation is retrieved. With the retrieved context activation, the sequence can then be predicted, i.e. generating $\mathbf{x}_{t+1}$, $\mathbf{x}_{t+2}$, etc., by using the learned weights and the updated potentials according to Equations (1), (2), (3), (4).

3.5.3. Early Stopping

I added an early stopping method as an optional part of the training procedure.
The entire set of teaching data is divided into training set and validation set. The training set is fed into the BPTT that updates the weights and initial context potentials, and the validation set is used for early stopping. Given a set of sequences as teaching data, every third sample of a sequence was excluded from training and used for validation, yielding a data division of 67 % for training and 33 % for validation. For each epoch of training, the early stopping method does forward propagation and computes the MSE on the validation set. The method keeps a history of the validation MSE together with the weights and initial context potentials for the latest Δh epochs where Δh is the size of the epoch window. During the training process, if the MSE on the training set becomes smaller than a defined minimum value (0.0009), the

method computes the gradient g_v of the validation MSE according to

$$g_v = \frac{\Delta e_v}{\Delta h} = \frac{e_v(h) - e_v(h - \Delta h)}{\Delta h} \tag{25}$$

where $e_v(h)$ is the validation MSE at the current epoch h. The value of Δh was set to 500. During the training process, the validation MSE declines together with the training MSE. However, if the validation MSE starts to rise ($g_v > 0$, which means $\Delta e_v > 0$), then the training is stopped and the weights and initial context potentials that correspond to the minimum validation MSE are returned.

3.5.4. Input-Output Preprocessing

By input-output preprocessing, I refer to preprocessing the input and postprocessing the output. In the default configuration, each dimension of an input sample is mapped to one IO neuron. For example, if the network is supposed to learn eight-dimensional sequences, then the number of IO neurons will be eight. An alternative configuration is an input-output preprocessing where each dimension of an input sample at time step t is mapped to more than one IO neuron. Likewise, after the network computed the prediction at $t + 1$, the activation of a number of IO neurons is mapped back to one dimension of the output sample. For a given input sample $\mathbf{x}$ with the dimension m, input mapping is done by $\mathbf{X} = \mathbf{x} \cdot \mathbf{v}^T$ where $\mathbf{v}$ is the preprocessing weight vector with the dimension n. The preprocessing vector has the elements $v_k \in \mathbb{R} \mid 0.0 < v_k < 1.0$. Thus, the number of IO neurons required is $m \cdot n$. Backward mapping, i.e. postprocessing, is done by $(\sum_k (X_{ik}/v_k))/dim(\mathbf{v})$ for each i where X_{ik} is an element of $\mathbf{X}$, v_k is an element of $\mathbf{v}$, with i as dimension index of the input pattern and k is the dimension index of the preprocessing weight vector. Unless defined otherwise, I use $\mathbf{v}^T = (0.225 \quad 0.45 \quad 0.9 \quad 0.45 \quad 0.225)$, yielding a pyramid-like activation pattern over each cluster of n IO neurons. The values of $\mathbf{v}$ were determined empirically and they can be different. When applying $\mathbf{v}$ to the input sample by using the aforementioned input mapping, the IO neurons have a sparse activation. This means that there are only a few neurons that have a high activation value (i.e. greater than 0.5), while all other adjacent neurons have significantly smaller values. In order to achieve this, the middle element v_M of $\mathbf{v}$ should have a high value close to 1.0 (i.e. $0.9 \leq v_M < 1.0$) and the values of the adjacent elements should be significantly smaller. The resulting sparse activation of the IO neurons improves the learning.

3.6. EO-MTRNN — Part 2: Autonomous Hyperparameter Estimation

In this section, I describe the extension of the modified MTRNN (Section 3.5) by the evolutionary optimization method SA-DE, resulting in the proposed EO-MTRNN. After visualizing the structure of the EO-MTRNN (Section 3.6.1), I explain the computation of the fitness value (Section 3.6.2). Finally, I describe my proposed optimization algorithm (Section 3.6.3) that computes the key hyperparameters of the network.

3.6.1. Structure of the Evolutionary Optimized MTRNN

Figure 7 gives a general overview on my proposed system for the autonomous estimation of the hyperparameters of a spatiotemporal learner. Besides a given set of teaching data, my proposed system consists of three main mechanisms:

- Spatiotemporal learner that is the modified MTRNN (Section 3.5)
- Evaluation metric that is the computation of a fitness value
- Optimizer (an evolutionary algorithm) that is the SA-DE [28]

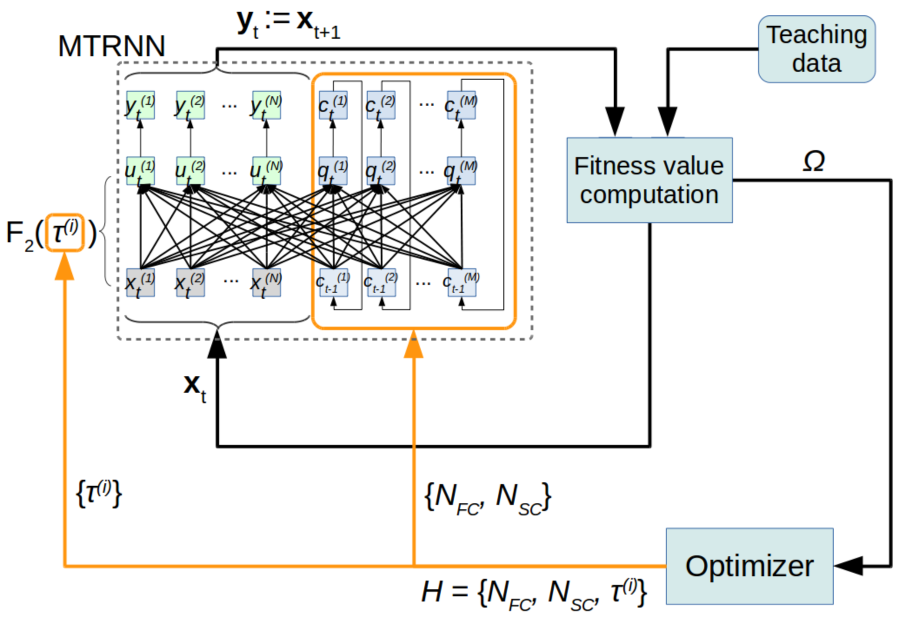


Figure 7 Proposed spatiotemporal learner capable of AHE: the evolutionary optimized MTRNN (EO-MTRNN). The spatiotemporal learner is the modified MTRNN (within the dashed box on the top left). The aim is to improve the learner performance by maximizing a fitness value Ω. For this purpose, the MTRNN is trained and tested with benchmark sequences (*Teaching data*). The fitness Ω is computed and fed into the *Optimizer* that estimates a set H of hyperparameters (orange colour) and uses H to restructure and retrain the MTRNN. A generation of the network is represented by H, its weights $\mathbf{W}$, and initial context potentials $\mathbf{q}^*$. For each generation, the *Optimizer* trains the MTRNN with the current population as well as with a mutated population of hyperparameters. For each individual of the population, a mutated individual of H will replace a population individual, if the mutated individual has a higher fitness value. Through this process, the *Optimizer* creates a new generation of the network that has a higher fitness, i.e. better performance, than the previous one. Here, H consists of the number of context neurons N_{FC}, N_{SC}, and the multiple timescales $\tau^{(i)}$ with $i \in IO, FC, SC$.

Formally, the working principle of the EO-MTRNN can be described by

$$\underset{\tau,\mathbf{n}}{\operatorname{argmax}}\, \Omega(\tau, \mathbf{n}, \underset{\mathbf{W},\mathbf{q}^\star}{\operatorname{argmin}}\, E(\mathbf{W}, \mathbf{q}^\star)) \tag{26}$$

where τ, $\mathbf{n}$, and $\mathbf{q}^\star$ denotes the vector of all neural timescales, number of context neurons, and initial context potentials, respectively, and $\mathbf{W}$ denotes the matrix of connective weights; the function Ω is a fitness function that is optimized via evolutionary optimization, while the

error function E is optimized via BPTT described in Section 3.5.2.
In sum, the EO-MTRNN optimizes the following parameters:

- Connective weights
- Initial potentials of context neurons
- Neural timescales
- Number of context neurons

Among these parameters, the neural timescales and the number of context neurons are treated as hyperparameters. They define the neural dynamics and the structure of the network, before the connective weights and initial potentials are adjusted by BPTT. Note that due to the connectivity of the network, an optimization of the number of context neurons implicitly also optimizes the number of connective weights.

3.6.2. Fitness Value Computation

The goal of the fitness value computation is to establish a mapping Ω from a vector $\mathbf{p} \in \mathbb{N}^{D_p}$ to a real scalar value, mathematically speaking $\Omega : \mathbf{p} \rightarrow \mathbb{R}$. The vector $\mathbf{p}$ has the dimension D_p (i.e. problem dimension) and represents the current setting of hyperparameters that are used for the optimization process. In case of the MTRNN hyperparameters, the vector $\mathbf{p}$ contains integer values. Table 6 shows the MTRNN hyperparameters which are optimized dependent on the value of the problem dimension D_p. The default setting is $D_P = 5$, then the

Problem dimension D_P	**Hyperparameter** H
1	τ_{SC}
2	τ_{FC}, τ_{SC}
3	$\tau_{IO}, \tau_{FC}, \tau_{SC}$
4	$N_{SC}, \tau_{IO}, \tau_{FC}, \tau_{SC}$
5	$N_{FC}, N_{SC}, \tau_{IO}, \tau_{FC}, \tau_{SC}$

Table 6 Dimension of the optimization problem and the corresponding MTRNN hyperparameters optimized through SA-DE.

number of all context neurons and all timescales are optimized. The other settings ($D_P < 5$) are optional; they can be used to customize the network optimization, for example if the user only wants to optimize one or more particular hyperparameter(s), while the others stay fixed. For any particular setting of hyperparameters, the MTRNN is trained with a set of sequences (represented by the box *Teaching data* in Figure 7). Note that the teaching data have to contain sequences (i.e. a time series of vectors), but it does not matter where these sequences have their origin. It can be either the set of benchmark sequences or any other data set, e.g. sensory-motor data collected on a robot.

The mapping Ω represents an evaluation metric and is considered as fitness value for the hyperparameter optimization process. This fitness value is computed by using the training data and the current network output. I use the normalized sum of the entries of the $S \times B$ matrix of correlation coefficients in order to compute Ω according to

$$\Omega := \frac{1}{S \cdot B} \sum_{i=1}^{S} \sum_{j=1}^{B} r_{ij} \tag{27}$$

where S is the number of training sequences and B is the number of spatial dimensions of the sequences.
Each entry $r_{ij} \in \mathbb{R} \mid -1.0 \leq r_{ij} \leq 1.0$ is computed by

$$r_{ij} = \frac{\sum_{t=1}^{L} (a_t^{(ij)} - \bar{a}^{(ij)})(y_t^{(ij)} - \bar{y}^{(ij)})}{\sqrt{\sum_{t=1}^{L} (a_t^{(ij)} - \bar{a}^{(ij)})^2} \sqrt{\sum_{t=1}^{L} (y_t^{(ij)} - \bar{y}^{(ij)})^2}} \tag{28}$$

where $a_t^{(ij)} \in \mathbb{R} \mid 0.0 < a_t^{(ij)} < 1.0$ denotes the value of a training sample at time step t of sequence i and its spatial dimension j. Accordingly, $y_t^{(ij)} \in \mathbb{R} \mid 0.0 < y_t^{(ij)} < 1.0$ denotes the value of a predicted sample at time step t of sequence i and its spatial dimension j. The mean value of sequence i and its spatial dimension j is given by $\bar{a}^{(ij)}$ and $\bar{y}^{(ij)}$ for training and prediction, respectively. The values $y_t^{(ij)}$ are obtained by running the MTRNN in closed loop as indicated in Figure 7.

3.6.3. Optimizer

This section completes the development of the proposed self-improving ST-learner, following my concept (Figure 4(b)) and its implication (Figure 5(b)).
Given Ω as the optimization metric (Section 3.6.2), the optimizer computes a new set of hyperparameters H that is used to restructure the spatiotemporal learner (MTRNN). In this case, restructuring means altering the timescales that effect the dynamics and altering the number of context neurons, both fast and slow context.
To this end, I adopted the method of DE with autonomous meta-parameter[1] adaptation proposed in [28]. It extends the original DE [179] by making its meta-parameters self-adapting. According to [28], the DE finds a *global* optimum over continuous spaces. Thus, I applied SA-DE because it offers a global optimization as well as autonomous meta-parameter adaptation. Using an optimizer with autonomous meta-parameter adaptation is beneficial, since the optimal meta-parameter settings are problem dependent [28].
The goal of the optimization is to maximize Ω. The optimization process works with individuals of two types of populations: the original population and the crossover population. Following the notation in Section 3.6.2, an individual i of the original population is described by the vector $\mathbf{p}_i$ containing the hyperparameters. Additionally, an individual i of the crossover population is described by the vector $\mathbf{c}_i$. Both populations have the same size NP, thus i going from 1 to NP. For example, if $NP = 4$, then the population consists of $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4\}$. A property

[1] Meta-parameters are also termed control parameters; they are the parameters of an optimization method that in turn is used to optimize the parameters of a target system.

of the SA-DE method is that $NP \geq 4$. The population size is initialized only once and kept constant during the optimization process, I used $NP = 4$. The SA-DE method contains two meta-parameters that are self-adapting: the mutation control $F \in \mathbb{R}$, also called differential weight, and the crossover control $CR \in \mathbb{R}$. The adaptation of F is influenced by the lower bound F_l and the upper bound F_u, the adaptation of CR is influenced by the probabilities τ_1 and τ_2. The parameters F_l, F_u, τ_1, and τ_2 are initialized only once and kept constant during the optimization process. I used the same values as in [28]: $F_l = 0.1$, $F_u = 0.9$, $\tau_1 = 0.1$, and $\tau_2 = 0.1$.

3.6.3.1 Computational Steps

The optimization process includes the following key steps:

1. *Initialize population*
   The first step of the algorithm is to initialize the original population with random positions in the search space. The search space is bounded by the vectors lower bound $\mathbf{b}_{lo}$ and upper bound $\mathbf{b}_{up}$, both of the problem dimension D_P. The control parameters F and CR are extended to become vectors $\mathbf{f}$ and $\mathbf{cr}$, respectively, both with dimension NP. Each element of $\mathbf{f}$ is initialized with a random floating-point value between F_l and F_u, meaning $f_i \in [F_l, F_u]$, from uniform distribution. Each element of $\mathbf{cr}$ is initialized with a random floating-point value between 0 and 1, meaning $cr_i \in [0, 1]$, from uniform distribution.

2. *Compute crossover individuals*
   For each individual, four random values r_1, r_2, r_3, $r_4 \in \mathbb{R}$ are generated with $r \in [0, 1]$, from uniform distribution. Then, each element of the control parameter vectors is adapted according to the following rule:

$$f_i^{(g+1)} = \begin{cases} F_l + r_1 \cdot F_u, & \text{if } r_2 < \tau_1 \\ f_i^{(g)}, & \text{otherwise} \end{cases} \tag{29}$$

$$cr_i^{(g+1)} = \begin{cases} r_3, & \text{if } r_4 < \tau_2 \\ cr_i^{(g)}, & \text{otherwise} \end{cases} \tag{30}$$

   where g, F_l, F_u, f_i, and cr_i denote the current generation, lower bound, upper bound, mutation control, and crossover control, respectively. The parameters τ_1 and τ_2 are probabilities for modifying f_i and cr_i, respectively. This adaptation of control parameters according to Equations (29), (30) is done *before* the next two steps, the mutation and the crossover.
   The mutation step and the crossover step are done for each individual i of the current population:
   The mutation step begins with picking three individuals a, b, c from the popula-

tion at random, these three have to be different from each other and different from the current target individual i that is being updated, this means mathematically $i \neq a \neq b \neq c$. A mutated individual $\mathbf{m}_i$, with dimension D_P, is computed by

$$\mathbf{m}_i = \mathbf{p}_a + f_i \cdot (\mathbf{p}_b - \mathbf{p}_c) \tag{31}$$

with population individuals $\mathbf{p}_a$, $\mathbf{p}_b$, $\mathbf{p}_c$, and crossover control f_i. The mutation $\mathbf{m}_i$ is then pruned to be within the search space bounded by $\mathbf{b}_{lo}$ and $\mathbf{b}_{up}$.
The crossover step begins with taking a random index d_x in the problem dimension, that means d_x is an integer with $d_x \in [1, ..., D_P]$. Then the crossover individual is computed, which is a mixture of an original individual and the mutated individual. For each dimension $k \in [1, ..., D_P]$, a uniform random (floating-point) number r_k is generated with $r_k \in [0, 1]$ and the vector element c_{ik} of the crossover individual $\mathbf{c}_i$ is computed according to

$$c_{ik} = \begin{cases} m_{ik}, & \text{if } r_k \leq cr_i \text{ or } k = d_x \\ p_{ik}, & \text{if } r_k > cr_i \text{ and } k \neq d_x \end{cases} \tag{32}$$

with m_{ik} as k-th element of the mutated individual $\mathbf{m}_i$ and p_{ik} as k-th element of the population individual $\mathbf{p}_i$.

3. *Compute next generation*
Based on the result of the previous computation of crossover individuals, this step implements a selection process that leads to a new population generation. In other words, it is the creation of a population $g + 1$ based on the previous population g.
The scalars $v_i^{(P)} \in \mathbb{R}$ and $v_i^{(C)} \in \mathbb{R}$ denote elements of the population fitness vector $\mathbf{v}^{(P)}$ and crossover fitness vector $\mathbf{v}^{(C)}$, respectively. Both vectors have the dimension NP. The computation of a fitness value v has been described in Section 3.6.2. Computation of $v_i^{(P)}$ requires a training of the MTRNN with the hyperparameters contained in the population individual $\mathbf{p}_i$. Computation of $v_i^{(C)}$ requires a training of the MTRNN with the hyperparameters contained in the crossover individual $\mathbf{c}_i$.
DE originally minimizes a fitness Ω. Maximization is done by setting $\Omega^* := -\Omega$ and using the fitness Ω^* when computing the selection.
Here, this is implemented by the boolean variable *max* that was always *true*, since the goal is fitness maximization. For each individual i, the update is

$$\left.\begin{aligned} v_i^{(P)} &\leftarrow -v_i^{(P)} \\ v_i^{(C)} &\leftarrow -v_i^{(C)} \end{aligned}\right\} \text{only if } \textit{max} = \textit{true}. \tag{33}$$

Then, for each individual i, the selection is done by

$$\mathbf{p}_i^{(g+1)} = \begin{cases} \mathbf{c}_i, & \text{if } v_i^{(C)} < v_i^{(P)} \\ \mathbf{p}_i^{(g)}, & \text{otherwise.} \end{cases} \tag{34}$$

In sum, after the initialization of the population (Step 1), the process updates the current population $\mathbf{p}$ consisting of NP individuals and updates their fitness $\mathbf{v}^{(P)}$. This update loop (containing Step 2 and Step 3) is executed for a given number of generations N_G.
Then, in the final step, the individual $\mathbf{p}_{sol}$ is returned that has the optimal fitness value $v_{opt}^{(P)}$. The value $v_{opt}^{(P)}$ is either a minimum fitness in case of minimization or a maximum fitness in case of maximization. Formally, this is done by setting

$$v_{opt}^{(P)} = v_1^{(P)}. \tag{35}$$

Then, for each individual i, do:

$$\left.\begin{aligned} v_{opt}^{(P)} &= v_i^{(P)} \\ sol &= i \end{aligned}\right\} \text{only if } v_i^{(P)} < v_{opt}^{(P)}. \tag{36}$$

The corresponding optimal fitness value is returned depending on the boolean variable *max* specifying minimization or maximization; here, it is maximization.

$$v_{opt}^{(P)} \leftarrow -v_{opt}^{(P)} \quad \text{only if } \textit{max} = \textit{true} \tag{37}$$

The output of the algorithm contains two parts: The first is $\mathbf{h}_{sol} := \mathbf{p}_{sol}$ containing the optimal MTRNN hyperparameters. The second is the corresponding fitness value $v_{opt}^{(P)}$.
All these aforementioned steps are integrated in Algorithm 1 that implements my proposed EO-MTRNN.

Algorithm 1 Optimization of MTRNN hyperparameters. Comments are in curly brackets.

Input: fitness value Ω (updated through retraining and re-evaluation of the learner)
Output: final solution vector $\mathbf{h}_{sol}$ of hyperparameters
Parameters: problem dimension D_P, population size NP, number of generations N_G, lower bound vector $\mathbf{b}_{lo}$, upper bound vector $\mathbf{b}_{up}$
Variables: generation index g, individual index i, population individual $\mathbf{p}_i$, crossover individual $\mathbf{c}_i$, fitness value $v_i^{(P)}$ of population individual $\mathbf{p}_i$, fitness value $v_i^{(C)}$ of crossover individual $\mathbf{c}_i$, vector $\mathbf{h}$ containing temporary hyperparameters

```
1:  Initialize population with $\mathbf{b}_{lo}$ and $\mathbf{b}_{up}$    {Step 1 in Section 3.6.3.1 }
2:  $g \leftarrow 0$
3:  while $g < N_G$ do
4:      Compute crossover individuals    {Step 2 in Section 3.6.3.1 }
5:      for $i \leftarrow 0; i < NP; i \leftarrow i + 1$ do
6:          Round $\mathbf{p}_i$
7:          $\mathbf{h} \leftarrow \mathbf{p}_i$
8:          Retrain MTRNN with $\mathbf{h}$
9:          Compute $\Omega$
10:         $v_i^{(P)} \leftarrow \Omega$
11:         Round $\mathbf{c}_i$
12:         $\mathbf{h} \leftarrow \mathbf{c}_i$
13:         Retrain MTRNN with $\mathbf{h}$
14:         Compute $\Omega$
15:         $v_i^{(C)} \leftarrow \Omega$
16:     end for
17:     Compute next generation using $\mathbf{v}^{(P)}$ and $\mathbf{v}^{(C)}$    {Step 3 in Section 3.6.3.1 }
18:     $g \leftarrow g + 1$
19: end while
20: Get $\mathbf{p}_{sol}$ with $v_{opt}^{(P)}$
21: $\mathbf{h}_{sol} \leftarrow \mathbf{p}_{sol}$
22: return $\mathbf{h}_{sol}$
```

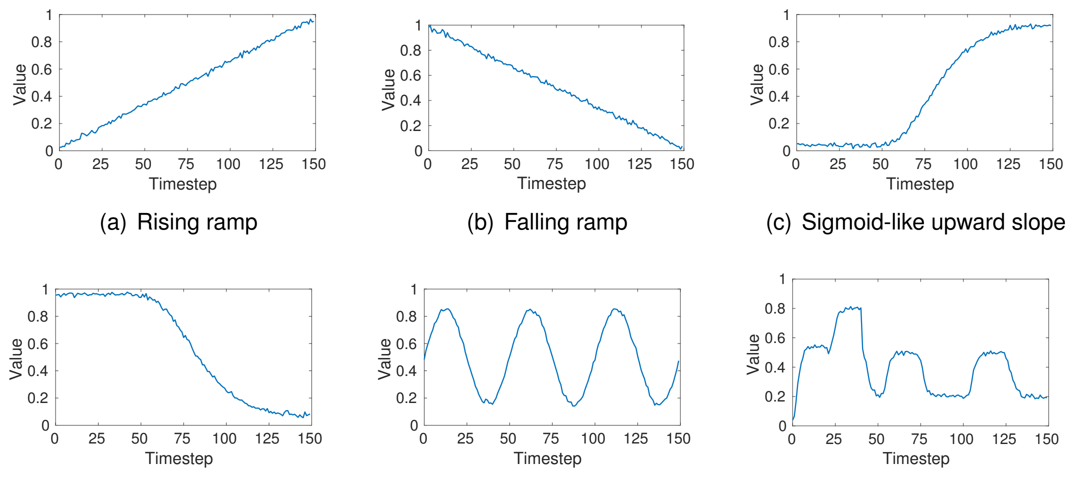
3.7. Benchmark Dataset for Empirical Analysis of Network Performance

In order to conduct an empirical analysis of the proposed EO-MTRNN, I propose a benchmark dataset that is used to evaluate the network performance in different configuration modes. Training the network on this benchmark dataset should yield an insight how the network performs depending on various factors such as the type of the training sequence, its dimension, and its length.

The benchmark training dataset is divided into sequences with one spatial dimension (elementary sequences) and sequences with multiple spatial dimensions. The multi-dimensional benchmark sequences are composed of the elementary sequences. Each spatial dimension of a benchmark sequence can be described mathematically, where y is the sample value and t is the discrete time step. Gaussian noise η was added with a mean $\mu = 0$ and variance $\sigma = 0.01$. This addition of noise simulates the property of teaching or training data which would be collected in a real application scenario, for example when sensory-motor training data would be collected on a robot through kinesthetic teaching. It is expected that the network should be robust to noise in the training data to a certain extent.

3.7.1. One-Dimensional Sequences

Table 7 provides a mathematical description of the elementary benchmark sequences. They are shown in Figure 8. Note that for the sine-like sequence (Figure 8(e)), I used its mathemati-



(a) Rising ramp

(b) Falling ramp

(c) Sigmoid-like upward slope

(d) Sigmoid-like downward slope

(e) Sine-like

(f) Irregular (type K)

Figure 8 Elementary benchmark training sequences of length $L = 150$. See Table 7 for their mathematical description used to generate them.

cal description in Table 7, incremented t by 0.02 to generate the function values, and assigned discrete time steps to each of those values separately. Since training data are often irregular in practice, i.e. they do not follow a regular shape like a ramp for example, I added training sequences of irregular type to the benchmark data set. These irregular sequences can be mathematically approximated by Gompertz functions, see Table 8. The irregular benchmark

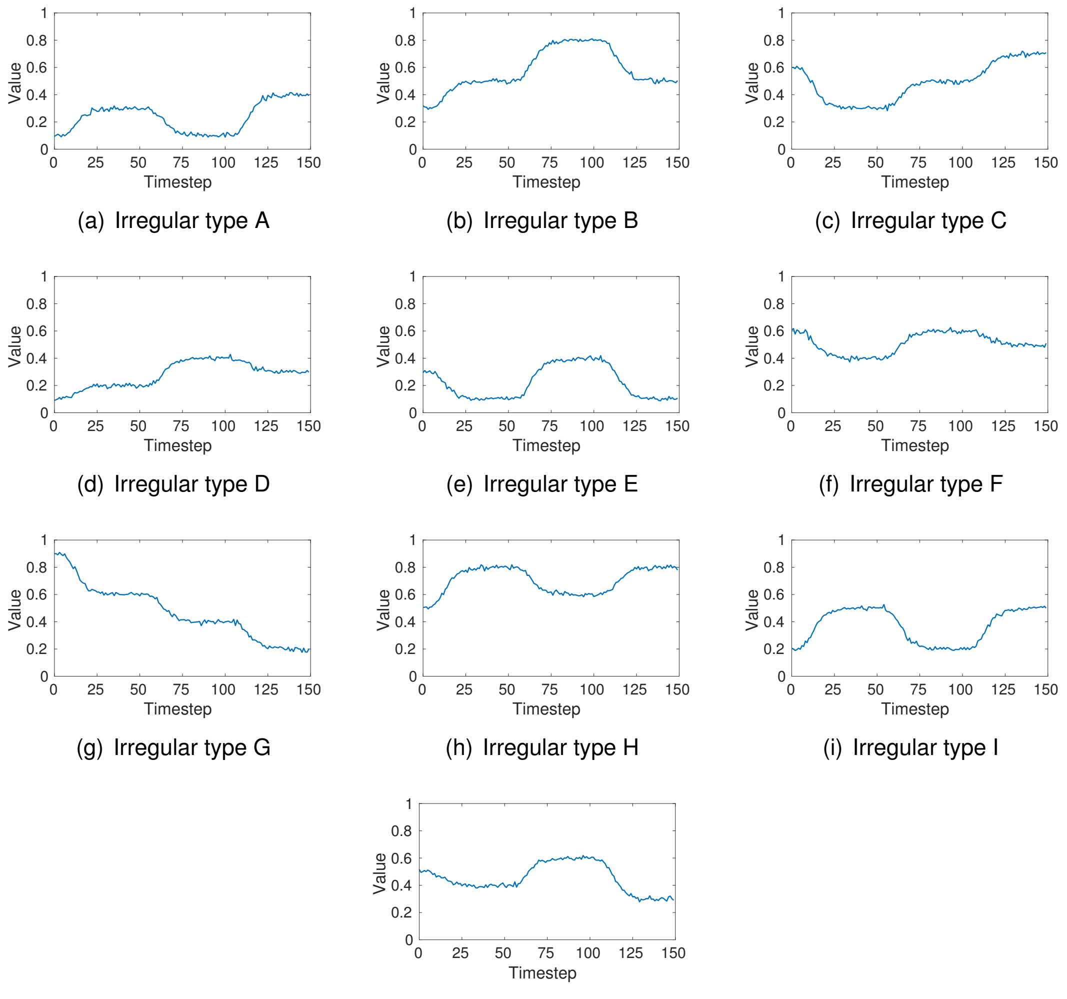
Type	**Length** L	**Mathematical expression** $y(t)$
Rising ramp	50	$y_1 = 0.0196t + 0.02 + \eta$
	100	$y_2 = 0.0097t + 0.02 + \eta$
	150	$y_3 = 0.0064t + 0.02 + \eta$
Falling ramp	50	$y_4 = -0.0196t + 0.98 + \eta$
	100	$y_5 = -0.0097t + 0.98 + \eta$
	150	$y_6 = -0.0064t + 0.98 + \eta$
Sigmoid-like upward slope	50	$y_7 = 0.9\exp(-10\exp(-0.2t)) + 0.04 + \eta$
	100	$y_8 = 0.9\exp(-50\exp(-0.1t)) + 0.04 + \eta$
	150	$y_9 = 0.9\exp(-100\exp(-0.06t)) + 0.04 + \eta$
Sigmoid-like downward slope	50	$y_{10} = -0.9\exp(-10\exp(-0.2t)) + 0.94 + \eta$
	100	$y_{11} = -0.9\exp(-50\exp(-0.1t)) + 0.96 + \eta$
	150	$y_{12} = -0.9\exp(-100\exp(-0.06t)) + 0.96 + \eta$
Sine	50	$y_{13} = 0.35\sin(2\pi t) + 0.5 + \eta$
	100	$y_{14} = 0.35\sin(2\pi t) + 0.5 + \eta$
	150	$y_{15} = 0.35\sin(2\pi t) + 0.5 + \eta$
Irregular (type K)	50	$0 \le t \le 20$: $y_{16} = 0.5\exp(-5\exp(-0.6t)) + 0.04 + \eta$
		$21 \le t \le 40$: $y_{16} = 0.3\exp(-5\exp(-0.6(t-21)) + 0.5 + \eta$
		$41 \le t < 50$: $y_{16} = -0.3\exp(-5\exp(-0.6(t-41))) + 0.5 + \eta$
	100	$0 \le t \le 50$: same as irregular (type K) with $L = 50$
		$51 \le t \le 70$: $y_{17} = 0.3\exp(-10\exp(-0.6(t-51))) + 0.2 + \eta$
		$71 \le t < 100$: $y_{17} = -0.3\exp(-19\exp(-0.6(t-71))) + 0.5 + \eta$
	150	$0 \le t \le 100$: same as irregular (type K) with $L = 100$
		$101 \le t \le 120$: $y_{18} = 0.3\exp(-12\exp(-0.6(t-101))) + 0.2 + \eta$
		$121 \le t < 150$: $y_{18} = -0.3\exp(-12\exp(-0.4(t-121))) + 0.5 + \eta$

Table 7 Elementary benchmark training sequences with Gaussian noise η with $\mu = 0$ and $\sigma = 0.01$.

Type	**Length** L	**Interval**	a	d	e
A	50	$0 \leq t < 50$:	0.2	0	0.1
	100	$51 \leq t < 100$:	-0.2	51	0.3
	150	$101 \leq t < 150$:	0.3	101	0.1
B	50	$0 \leq t < 50$:	0.2	0	0.3
	100	$51 \leq t < 100$:	0.3	51	0.5
	150	$101 \leq t < 150$:	-0.3	101	0.8
C	50	$0 \leq t < 50$:	-0.3	0	0.6
	100	$51 \leq t < 100$:	0.2	51	0.3
	150	$101 \leq t < 150$:	0.2	101	0.5
D	50	$0 \leq t < 50$:	0.1	0	0.1
	100	$51 \leq t < 100$:	0.2	51	0.2
	150	$101 \leq t < 150$:	-0.1	101	0.4
E	50	$0 \leq t < 50$:	-0.2	0	0.3
	100	$51 \leq t < 100$:	0.3	51	0.1
	150	$101 \leq t < 150$:	-0.3	101	0.4
F	50	$0 \leq t < 50$:	-0.2	0	0.6
	100	$51 \leq t < 100$:	0.2	51	0.4
	150	$101 \leq t < 150$:	-0.1	101	0.6
G	50	$0 \leq t < 50$:	-0.3	0	0.9
	100	$51 \leq t < 100$:	-0.2	51	0.6
	150	$101 \leq t < 150$:	-0.2	101	0.4
H	50	$0 \leq t < 50$:	0.3	0	0.5
	100	$51 \leq t < 100$:	-0.2	51	0.8
	150	$101 \leq t < 150$:	0.2	101	0.6
I	50	$0 \leq t < 50$:	0.3	0	0.2
	100	$51 \leq t < 100$:	-0.3	51	0.5
	150	$101 \leq t < 150$:	0.3	101	0.2
J	50	$0 \leq t < 50$:	-0.1	0	0.5
	100	$51 \leq t < 100$:	0.2	51	0.4
	150	$101 \leq t < 150$:	-0.3	101	0.6

Table 8 Elementary benchmark training sequences of irregular type. Each sequence type can be described by $y(t) = a\exp(-b\exp(-c(t-d))) + e + \eta$, with $b = 10$, $c = 0.2$, and η is Gaussian noise with constants $\mu = 0$ and $\sigma = 0.01$. The values of the remaining parameters a, d, and e are shown in columns 4 to 6.

sequences are shown in Figure 9.



(a) Irregular type A (b) Irregular type B (c) Irregular type C

(d) Irregular type D (e) Irregular type E (f) Irregular type F

(g) Irregular type G (h) Irregular type H (i) Irregular type I

(j) Irregular type J

Figure 9 Elementary benchmark training sequences of irregular type. See Table 8 for their mathematical description.

3.7.2. Multi-Dimensional Sequences

In practice, training sequences provided to a recurrent neural network typically have multiple spatial dimensions. For my benchmark data set, each spatial dimension contains one of the elementary benchmark training sequences from Table 8. The multi-dimensional benchmark sequences are described in Table 9.

3.7.3. Network Parameterization

For the learning of both one-dimensional and multi-dimensional sequences (Section 3.7.1 and Section 3.7.2), I used the following parameterization in Table 10. Note that the variable number of IO neurons in Table 10 results from the given network configuration mode. For example, a four-dimensional input pattern requires exactly four IO neurons if the preprocessing is de-activated (one-to-one mapping). The same pattern requires 20 IO neurons if preprocessing is activated (four-dimensional input times five-dimensional preprocessing weight

Spatial dimensions	**Elementary type**
1	D
2	D, E
4	D, E, C, B
6	D, E, C, B, A, F
8	D, E, C, B, A, F, H, G
10	D, E, C, B, A, F, H, G, J, I

Table 9 Multi-dimensional benchmark training sequences. See Table 8 for each elementary type. For example, the 2-dimensional sequence consists of type D as its first spatial dimension and type E as its second spatial dimension. Note that the order does not matter, it can be composed randomly.

N_{IO}	N_{FC}	N_{SC}	τ_{IO}	τ_{FC}	τ_{SC}
variable	20	5	20	25	250

Table 10 Number of neurons and timescales for the learning presented in Sections 3.7.1 and 3.7.2.

vector). The values shown in Table 10 were kept fixed, until I proceeded with the hyperparameter estimation experiments that are presented later in this chapter.
The values for the learning rates and momentum are summarized in Table 11. These values for the learning rates and momentum were kept fixed throughout all experiments, also during the hyperparameter estimation (that focused on estimating the number of neurons and all timescales).

MSE_{T}	α	β_{FC}	β_{SC}	η
≥ 0.03	0.6	0.6	0.6	0.9
< 0.03	0.4	0.4	0.4	0.9

Table 11 Learning rates α, β_{FC}, β_{SC} and momentum η kept fixed for all experiments.

3.7.4. Termination Criterion for BPTT

In each training session, the BPTT terminated if the number of epochs reached $5.0 \cdot 10^5$ or if the training MSE reached $3.0 \cdot 10^{-5}$, independent of the network configuration. In case of early stopping, the BPTT also stopped if the training MSE was below $9.0 \cdot 10^{-4}$ and the validation MSE started to rise.

3.8. Results: Validation of the EO-MTRNN

This section presents the results obtained from various experiments: Firstly, I present the validation results of the MTRNN block that was trained with the proposed benchmark training dataset. These results are presented in the Sections 3.8.3 and 3.8.4. Then, in Section 3.8.5, I empirically prove the correctness of my implementation of the SA-DE method. In Section 3.8.6, I present the validation results of the EO-MTRNN trained with the proposed benchmark dataset. Finally, in Section 3.8.7, I show the validation results of the EO-MTRNN trained with real sensory-motor data from a robot experiment.

3.8.1. Evaluation Metric for the Learning Capability

The learning capability was investigated depending on the length of the benchmark training sequences, the type of the training sequences, and their dimension. The learning capability is measured by Ω defined in Equation (27). In the following, this metric is referred to as R-value with $R \in \mathbb{R} \mid -1.0 \leq R \leq 1.0$, since it is a normalized sum of correlation coefficients. Each correlation coefficient describes how well the prediction fits the observed data.

3.8.2. MTRNN Configuration Modes

For the proposed network, I evaluated the four configuration modes that result from the proposed changes to the network:

- preprocessing *off* and early stopping *off*
- preprocessing *off* and early stopping *on*
- preprocessing *on* and early stopping *off*
- preprocessing *on* and early stopping *on*

Note that an enabled preprocessing of input data (i.e. preprocessing *on*) implicitly includes a postprocessing of the network output.

3.8.3. Results of Learning One-Dimensional Sequences

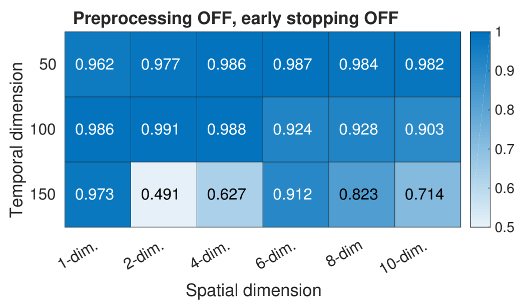
Table 12 compares the four different configuration modes of the EO-MTRNN. Each configuration mode was evaluated by learning different one-dimensional benchmark sequences.

3.8.4. Results of Learning Multi-Dimensional Sequences

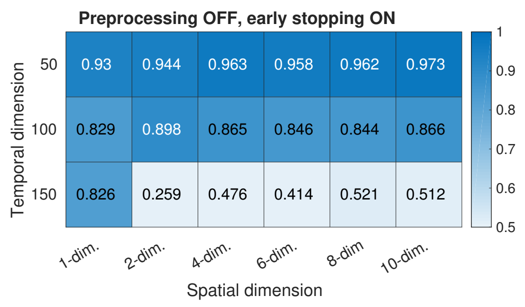
The network is trained with multi-dimensional sequences (Table 9). The learning results are summarized in Figure 10. An example case of learning and recall is shown in Figure 11. An example of the extrapolation behaviour of the net, i.e. the prediction of a sequence over a much longer timespan than the original timespan given in training, is shown in Figure 12.

Length	**Pre-processing**	**Early stopping**	**Falling ramp**	**Rising ramp**	**Sigmoid-like down-ward slope**	**Sigmoid-like upward slope**	**Sine**	**Irregular (type K)**
50	*off*	*off*	0.992	-0.0415	0.997	0.998	0.937	0.479
50	*off*	*on*	0.966	0.994	0.986	0.988	0.120	0.333
50	*on*	*off*	0.998	0.998	0.995	0.996	0.945	0.318
50	*on*	*on*	0.965	0.970	0.987	0.996	0.900	0.501
100	*off*	*off*	0.994	-0.447	0.999	1.00	0.827	0.451
100	*off*	*on*	0.953	0.982	0.943	0.944	0.285	0.0458
100	*on*	*off*	0.996	0.999	0.998	0.999	0.644	0.410
100	*on*	*on*	0.940	0.954	0.941	0.962	0.491	0.419
150	*off*	*off*	0.995	-0.404	0.999	0.999	0.664	0.177
150	*off*	*on*	0.930	0.950	0.901	0.905	-0.0408	0.106
150	*on*	*off*	-0.617	0.999	0.999	0.999	0.196	0.793
150	*on*	*on*	0.707	0.844	0.885	0.939	0.172	0.488

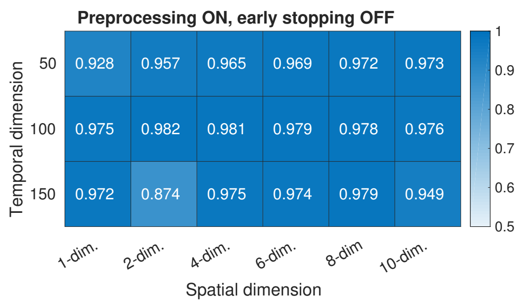
Table 12 Learning of one-dimensional benchmark sequences. The learning is measured by the R-value. Four different network configuration modes were compared (second and third column), given the length of a training sequence. The training sequences (fourth to ninth column) are shown in Figure 8.



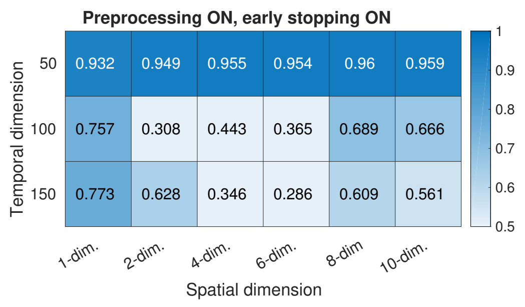
(a) Network configuration 1: preprocessing *off*, early stopping *off*



(b) Network configuration 2: preprocessing *off*, early stopping *on*

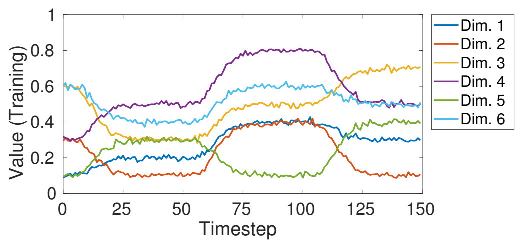


(c) Network configuration 3: preprocessing *on*, early stopping *off*

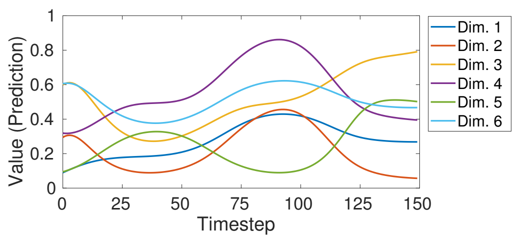


(d) Network configuration 4: preprocessing *on*, early stopping *on*

Figure 10 Learning of multi-dimensional benchmark training sequences. The learning is measured by the R-value (indicated by the colour bar). Four different network configuration modes were compared, given the length of a training sequence. The training sequences are encoded in Table 9.



(a) 6-dimensional benchmark training sequence



(b) Recall after learning

Figure 11 Learning and recall of the noisy 6-dimensional benchmark training sequence with length $L = 150$. Network configuration: preprocessing *on*, early stopping *off*. Achieved R-value: 0.974. See Table 9 for details on the training data.

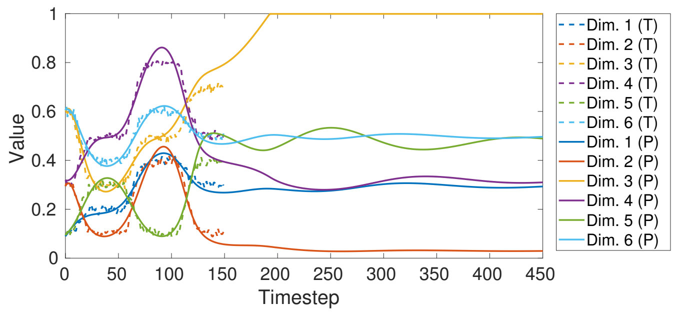


Figure 12 Recall with extrapolation from time step 150 to 450. Dashed lines are the training sequence labelled (T), see also Figure 11(a) above. Solid lines are the predicted sequence labelled (P). The network tends to extrapolate the sequence based on the latest history of input-output activations, behaving like a type of predictive memory.

3.8.5. Validation of the Implementation of the Optimization Method

In order to validate my implementation of the SA-DE [28], I compared my numerical results (obtained from my implementation) with the numerical results obtained from the original version in [28]. For the SA-DE, I used the same parameter settings as given in [28]. Table 13 shows that my results concur with [28], proving the correctness of my implementation. Especially the results on benchmark functions f_8, f_9, f_{10}, f_{11}, f_{12}, f_{13} are important to consider, since they demonstrate the ability of the method to find a global optimum despite a high number of local optima [28], [232], [200].

3.8.6. Improvement of Learning Capability by Autonomous Hyperparameter Estimation — Single Sequences and Multiple Sequences

Here, it is investigated whether my proposed Algorithm 1 can improve the learning of given training data. Algorithm 1 performs the AHE in order to yield an evolutionary optimized network. The problem dimension (see Table 6) was set to five, making the EO-MTRNN estimate the number of context neurons N_{FC}, N_{SC}, and the different timescales τ_{IO}, τ_{FC}, τ_{SC}. For the SA-DE, the population size was $NP = 4$ (minimum number possible due to the property of SA-DE) and the number of generations was $N_{Gen.} = 10$.

In contrast to the original DE, the SA-DE automatically adjusts the values for the crossover probability CR and the differential weight F. The values of F and CR were updated by Equations (29) and (30), respectively, with $F \in [0.1, 1.0]$ and $CR \in [0.0, 1.0]$.

Regarding the search space, the bounds for the context neurons were set to $5 \leq N \leq 30$ for fast context and slow context, respectively. The bounds for the timescales were set to $2 \leq \tau \leq 300$ for input-output, fast context, and slow context group, respectively.

For comparison purposes, I used a weak to reasonable default parameterization of the network, see Table 14.

Benchmark function f from [28]	**Number of generations**	**Own implementation of SA-DE**	**Original SA-DE [28]**
f_1	1500	$2.0 \cdot 10^{-16}$	$1.1 \cdot 10^{-28}$
	2400	$5.7 \cdot 10^{-29}$	
f_2	2000	$3.3 \cdot 10^{-14}$	$1.0 \cdot 10^{-23}$
	3200	$1.5 \cdot 10^{-23}$	
f_3	5000	$4.9 \cdot 10^{-3}$	$3.1 \cdot 10^{-14}$
	13000	$2.4 \cdot 10^{-14}$	
f_4	5000	$2.2 \cdot 10^{-9}$	0
f_5	20000	$2.9 \cdot 10^{-30}$	0
f_6	1500	0	0
f_7	3000	$2.83 \cdot 10^{-1}$	$3.15 \cdot 10^{-3}$
f_8	9000	-12569.5	-12569.5
f_9	5000	0	0
f_{10}	1500	$3.7 \cdot 10^{-9}$	$7.7 \cdot 10^{-15}$
	2500	$7.2 \cdot 10^{-15}$	
f_{11}	2000	0	0
f_{12}	1500	$1.7 \cdot 10^{-17}$	$6.6 \cdot 10^{-30}$
	2400	$6.4 \cdot 10^{-30}$	
f_{13}	1500	$1.1 \cdot 10^{-16}$	$5.0 \cdot 10^{-29}$
	2400	$5.3 \cdot 10^{-29}$	
f_{16}	100	-1.03163	-1.03163
f_{17}	100	0.397887	0.397887
f_{18}	100	3	3

Table 13 Validation of the evolutionary optimizer: Comparison of numerical results using the benchmark function set from [28]. The main results are the minima (columns 3 and 4) of particular benchmark functions, the minima are averaged over 50 independent runs. Note that the purpose of this comparison is to validate a correct implementation of the SA-DE optimization method. The concurring results (columns 3 and 4) indicate a correct implementation.

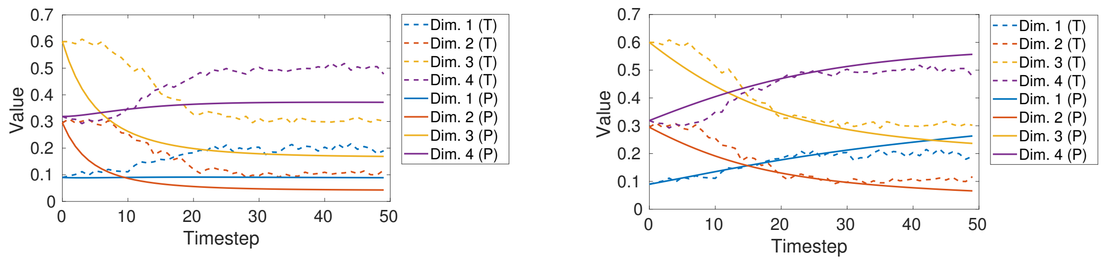
N_{IO}	N_{FC}	N_{SC}	τ_{IO}	τ_{FC}	τ_{SC}
variable	15	5	10	20	40

Table 14 Default parameterization for comparing the results of hyperparameter estimation shown in Section 3.8.6.

3.8.6.1 Single Sequences

The number of IO neurons was pre-determined by the given network configuration, it was the same as described in Section 3.7.3.

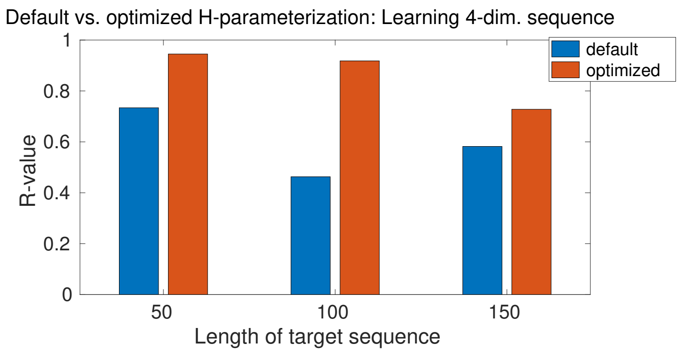
Figure 13 shows an example how the hyperparameter estimation improved the learning result. To summarize, a performance comparison between a default and an optimized parameterization is shown in Figure 14.



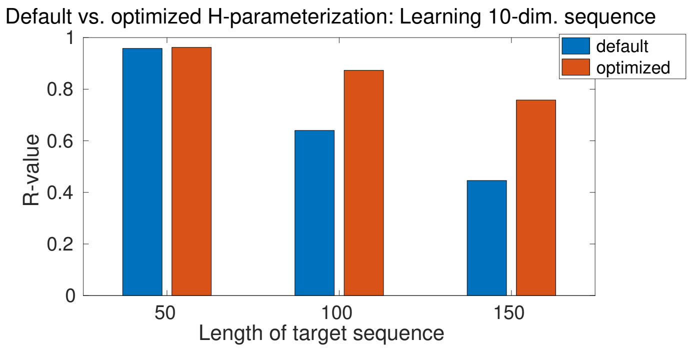
(a) Default parameterization with $N_{FC} = 15$, $N_{SC} = 5$, $\tau_{IO} = 10$, $\tau_{FC} = 20$, $\tau_{SC} = 40$. Achieved R-value: 0.734.

(b) Optimized parameterization with $N_{FC} = 14$, $N_{SC} = 30$, $\tau_{IO} = 142$, $\tau_{FC} = 37$, $\tau_{SC} = 201$, computed by Algorithm 1. Achieved R-value: 0.945. The prediction (P) shows a better fitting of the data (T) than in the default case above.

Figure 13 Example of hyperparameter estimation. The target sequence (dashed lines) had 4 spatial dimensions and length 50. The configuration was preprocessing on and early stopping on meaning that only 33 of 50 samples were used for training. The recall performance of a weak to reasonable default parameterization is shown in 13(a). Applying my proposed hyperparameter optimization to N_{FC}, N_{SC}, τ_{IO}, τ_{FC}, τ_{SC} increased the performance by 28.7 % without overfitting the data, see 13(b).



(a) Performance gain when learning a 4-dimensional target sequence. A concrete example is the case of length 50 shown in Figure 13. The gain is 28.7 %, 98.3 %, 25.1 % for the length 50, 100, 150, respectively.



(b) Performance gain when learning a 10-dimensional target sequence. The gain is 0.4 %, 36.4 %, 70.0 % for the length 50, 100, 150, respectively.

Figure 14 Default versus optimized hyperparameterization for single sequences. Comparison of learning performance when learning a multi-dimensional target sequence with different lengths, network configuration was preprocessing on and early stopping on. From these cases, it follows an average performance gain of approximately 43 % compared to the default parameterization given in Table 14.

3.8.6.2 Multiple Sequences

Using the proposed benchmark sequences of irregular type (Figure 9), I also validated different cases of *simultaneous* learning of multiple sequences. I simultaneously trained 7 sequences, each of them had 4 spatial dimensions. The assembly of these sequences is given in Table 15.

Sequence number	**Spatial composition**
1	D, E, C, B
2	A, F, H, G
3	J, I, D, E
4	B, C, E, D
5	E, A, J, B
6	I, J, H, A
7	F, G, D, C

Table 15 Benchmark training sequences of irregular type, all 7 sequences were trained simultaneously. Each spatial dimension contains an elementary sequence of irregular type (see Figure 9 for the visualization of each spatial dimension). The arrangement of the spatial dimensions does not matter, it was composed randomly.

Three different cases of temporal dimensions (i.e. sequence lengths) were evaluated: $L = 50$, $L = 100$, and $L = 150$. The optimized hyperparameters were computed by Algorithm 1 trained with 7 sequences simultaneously, each with $L = 150$. The obtained hyperparameters were then kept constant and used to train the network with different lengths per sequence: $L = 50$, $L = 100$, $L = 150$. Since 7 sequences were trained simultaneously for each case, the total number of samples was 350, 700, and 1050, respectively. Each case was also trained with default hyperparameters (Table 14) to compare the performance.

An 11-dimensional weight vector was used for the analytical pre- and postprocessor generating sparse representation of *IO* data. The middle element of this vector is close to 1.0, while the other elements are close to 0.0. Beginning with the first and going to the last, the elements of the weight vector $\mathbf{v}^T$ were 0.007812, 0.015625, 0.03125, 0.0625, 0.125, 0.99, 0.125, 0.0625, 0.03125, 0.015625, and 0.007812.

The computed hyperparameters were $N_{FC} = 10$, $N_{SC} = 9$, $\tau_{IO} = 19$, $\tau_{FC} = 167$, $\tau_{SC} = 37$. Looking at the timescales, the roles of the fast context and slow context neurons were swapped by the evolutionary algorithm, i.e. the fast context group became slow context and vice versa. Nevertheless, this can be justified by the results when comparing the learning performance with the default parameterization. The performance results are summarized in Figure 15.

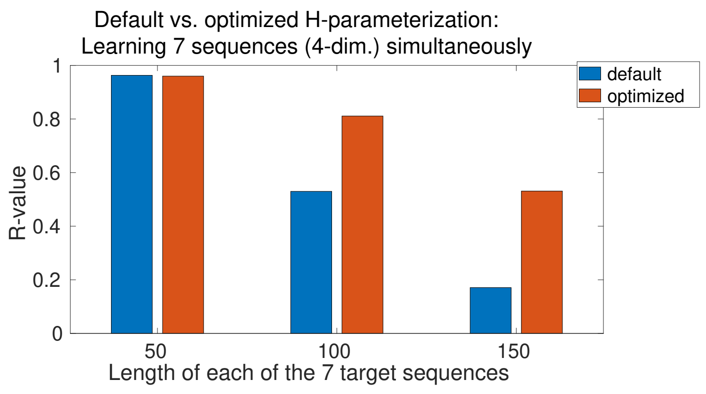
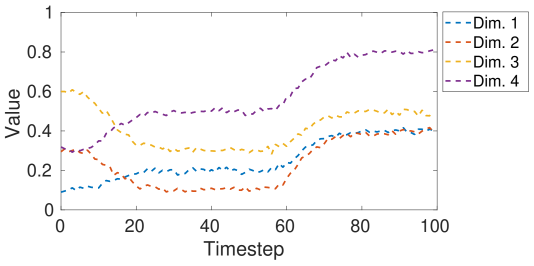
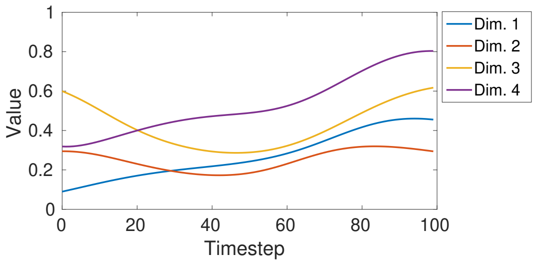


Figure 15 Default versus optimized hyperparameterization for multiple sequences trained simultaneously. Network configuration was preprocessing *on* and early stopping *off*. The greater the size of training data, the more the evolutionary optimization is worth it. Cases $L = 100$ and $L = 150$ have great differences in the R-value, respectively: 0.530 versus 0.811 (53.0 % gain) and 0.171 versus 0.531 (210.5 % gain, albeit poor performance in the default case). As an example, the optimized case with $L = 100$ is visualized by Figures 16 and 17.

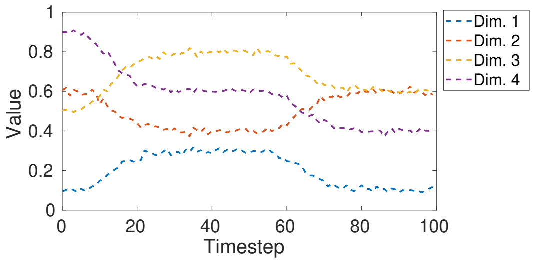
For each case of training with default and optimized hyperparameters, the number of epochs was 10^6. An example of sequence recall is shown in Figures 16 and 17, where 7 sequences (each $L = 100$) were trained simultaneously with the optimized hyperparameters.
Corresponding to this example, I also investigated the self-organization of the initial activation states of the context neurons. Figure 18 shows these initial activation states.
In order to find suitable hyperparameters for these benchmark sequences trained simultaneously, the number of epochs was set to 15000. This is relatively short but was necessary due to time restrictions. Using 15000 epochs per single individual of the network population going through the evolutionary process and using the learning rates in Table 11, Algorithm 1 takes roughly 35 hours to find suitable hyperparameters for 1050 (i.e. 7×150) sample vectors with 4 dimensions, when run on an i7-7500U central processing unit (CPU) (2.7 Ghz). Note that this can be significantly speeded up by running the proposed algorithm on a graphics processing unit (GPU) supporting parallel programming, e.g. CUDA. This is important to consider because many more epochs per training and more evolutionary generations can be computed within the same timeframe. Consequently, GPU usage would boost the learning performance within the same timeframe.



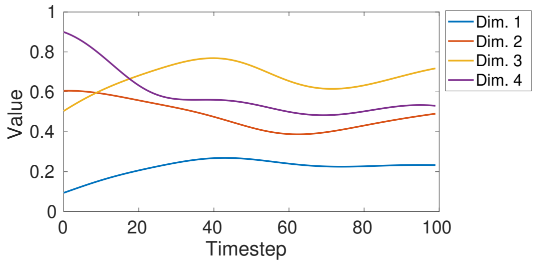
(a) Sequence 1



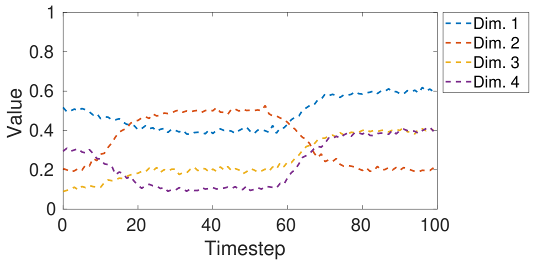
(b) Recall of sequence 1



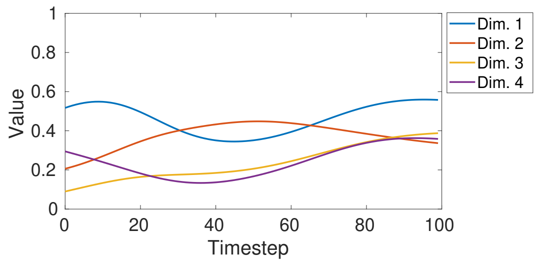
(c) Sequence 2



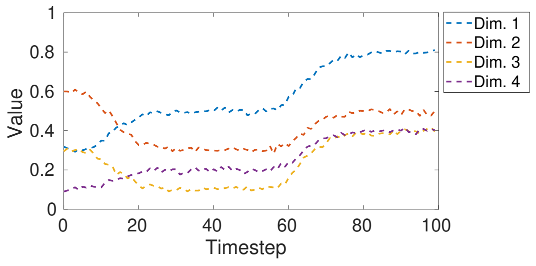
(d) Recall of sequence 2



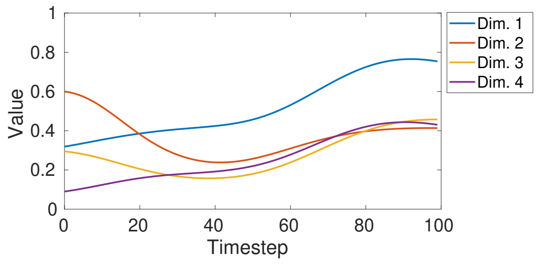
(e) Sequence 3



(f) Recall of sequence 3

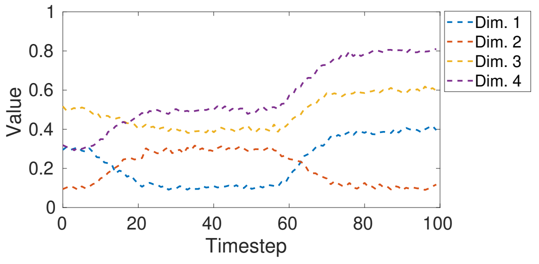


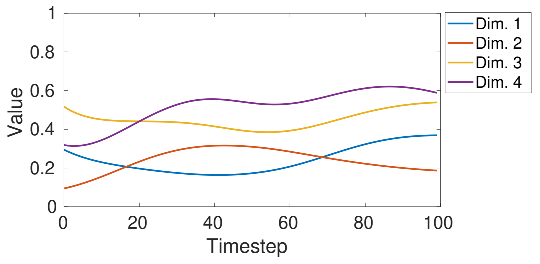
(g) Sequence 4

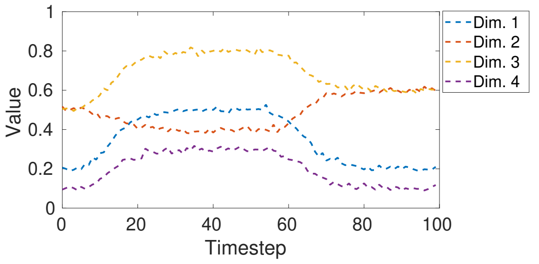


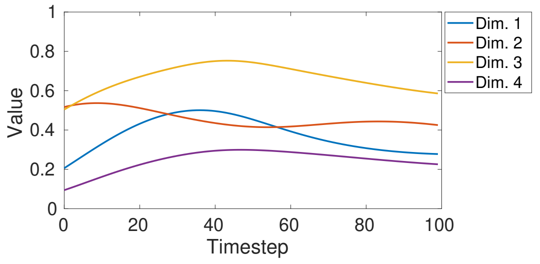
(h) Recall of sequence 4

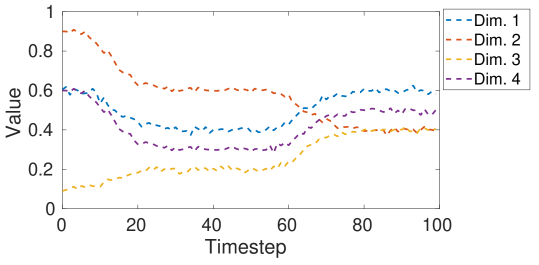
Figure 16 Simultaneous training of 7 benchmark sequences ($L = 100$) and their recall: Sequences 1 to 4. See Figure 17 for the sequences 5 to 7.


(a) Sequence 5


(b) Recall of sequence 5


(c) Sequence 6


(d) Recall of sequence 6


(e) Sequence 7

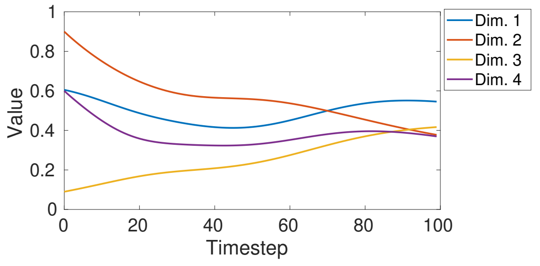
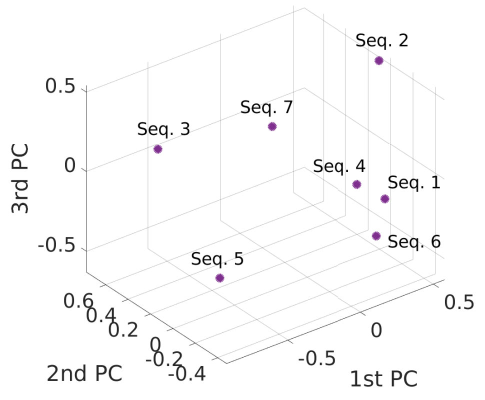
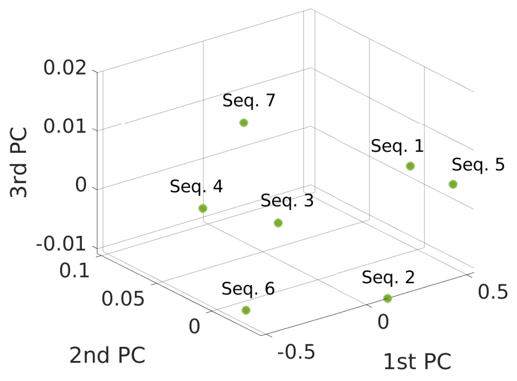

(f) Recall of sequence 7

Figure 17 Simultaneous training of 7 benchmark sequences ($L = 100$) and their recall: Sequences 5 to 7.



(a) Initial activation states of the fast context group



(b) Initial activation states of the slow context group

Figure 18 Initial activation states of the context group in principal component (PC) space after being trained with 7 sequences simultaneously (each sequence with $L = 100$), using optimized hyperparameters. It can be seen that all 7 sequences are clearly separable in the activation space of each context group. Mapped from each initial activation state of fast and slow context, the corresponding sequence can be recalled (Figures 16 and 17) by using the learned weights.

3.8.7. Performance with Robot Data

In this section, I investigated the performance of the proposed EO-MTRNN when it is trained with action sequences that were taught to a Sony QRIO humanoid robot in [230]. In the experiment conducted by Yamashita and Tani, the robot was fixed to a stand and manipulated a cubic object that was placed on a workbench in front of the robot. The action sequences taught to the robot consisted of several manipulation primitives using both arms, e.g. reach and grasp the object, and move the object up and down three times. Each action sequence begins and ends in a defined home position. The action sequences are sensory-motor sequences, each of them has 10 spatial dimensions but a different length depending on the interaction taught. Among the 10 spatial dimensions, the first 8 dimensions represent the DOF of the robot arm: 3 DOF shoulder and 1 DOF elbow per arm. The remaining 2 dimensions represent the horizontal (X) and vertical (Y) axis of vision, since the 2 DOF head joint followed the object automatically by a pre-given visual servoing. Each 10-dimensional sensory-motor pattern was sampled every 150 milliseconds.

3.8.7.1 EO-MTRNN: Learning Robot Sensory-Motor Data with Adverse Hyperparameters

In the following experiment, the objective is to investigate the mental simulation performance of the EO-MTRNN when deliberately operated with hyperparameters that are far from the optimum.

Using the teaching data from [230], a sequence was selected representing the following behaviour: starting from home position, reach for the object with both arms, grasp it, and move it up and down three times, then go back to home position. For comparison, this teaching sequence is partly shown in the first column ("Teach") of Figure 4 in [230]. The teaching sequence represents a manipulation task that is shown in Figure 19. The EO-MTRNN was

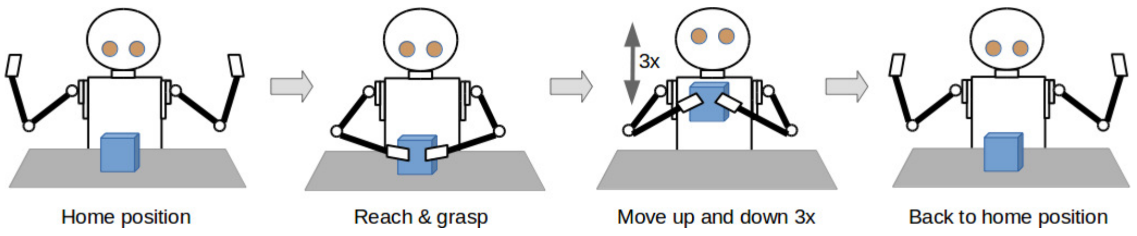


Figure 19 Robot task in [230] to obtain sensory-motor data through kinesthetic teaching. In home position, the robot is facing a box (blue) on a workbench (grey). It reaches and grasps the box. Then, the robot moves the box up and down three times, with its head cameras always focusing on the box by moving the head-neck joint accordingly. Finally, the robot returns back to home position.

trained with this sequence, using a default configuration without AHE (i.e. without the evolutionary optimization) in order to investigate a *worst-case* scenario.

The number of context neurons and the timescales were set as follows: $N_{FC} = 15$, $N_{SC} = 5$, $\tau_{IO} = 10$, $\tau_{FC} = 20$, $\tau_{SC} = 40$. This is a configuration that would be far from an optimal one, especially regarding the ratio of τ_{FC} and τ_{SC}. Nevertheless, the purpose here is to investigate how well the proposed network performs in case of adverse hyperparameters, i.e. whether it is still able to learn the data.

For the analytical pre- and postprocessor generating sparse representation of IO data, the same 11-dimensional weight vector as in Section 3.8.6.2 was used. With 8-dimensional proprioceptive and 2-dimensional visual data, this resulted in 88 neurons encoding the motor part of the IO group and 22 neurons encoding the visual part of the IO group. This yielded $N_{IO} = 110$.

When the proposed network is applied to learn sensory-motor data, the connective weights between the IO neurons encoding the sensory (i.e. visual) part and the IO neurons encoding the motor (i.e. proprioceptive) part are set to zero.

The proposed network reached an MSE of $3 \cdot 10^{-6}$ after 821990 epochs. I validated the recall of this sequence, i.e. its mental simulation. Figure 20 shows the results. The results show a

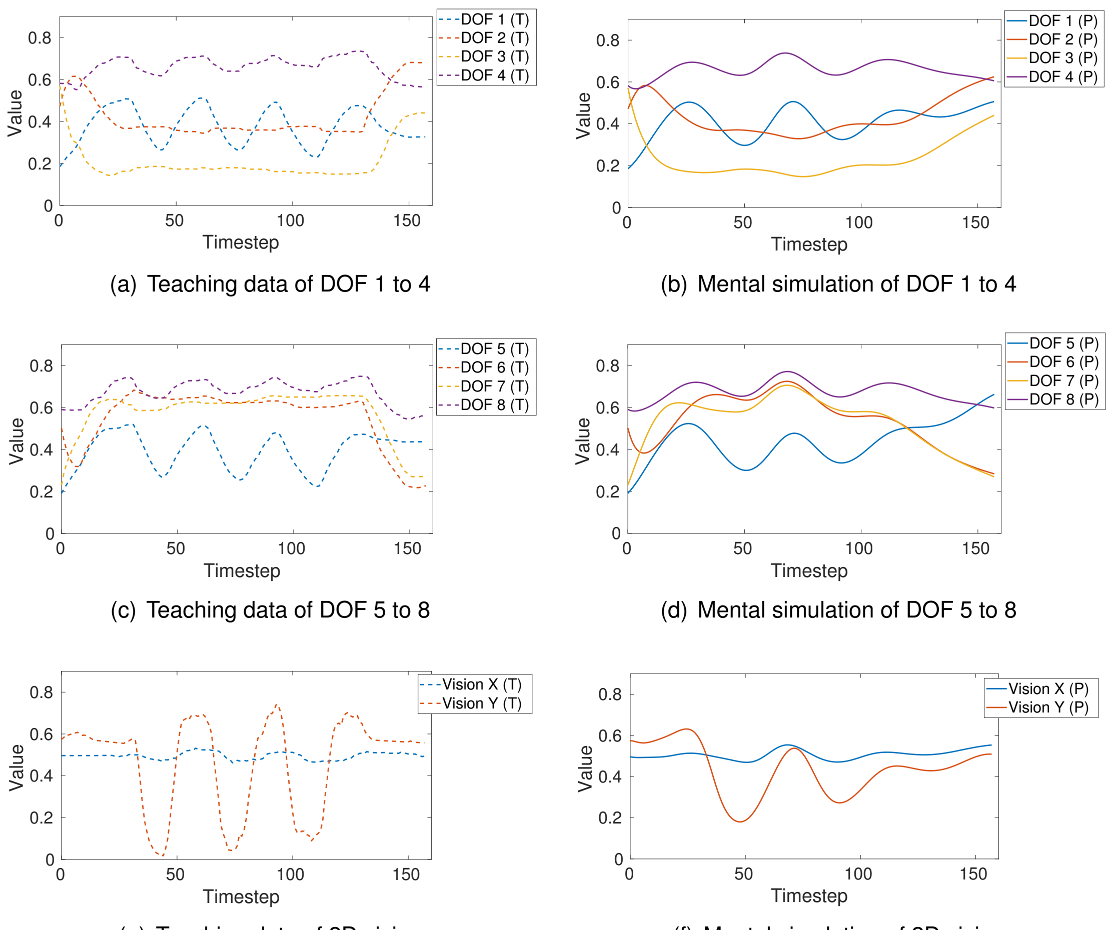


Figure 20 Performance of the proposed EO-MTRNN (with default parameterization) at learning robot sensory-motor data. Left side (20(a), 20(c), 20(e)): Teaching data of the behaviour sequence consisting of reaching and grasping the box, and moving it up-down as depicted in Figure 19. This teaching data originated from [230]. Right side (20(b), 20(d), 20(f)): Recall of the sequence (i.e. mental simulation) by the proposed network, with default (i.e. non-optimized) hyperparameters reflecting a *worst-case* scenario. The number of context neurons was rather small and the ratio of timescales was not chosen to be optimal in order to evaluate how a non-optimized configuration of the network performs, i.e. whether it can sufficiently learn the teaching data. Note again that the hyperparameters are *not* optimized in this case; here, it was of interest whether the proposed network can still preserve the task structure in case of a single learning procedure *without* going through the evolutionary optimization process.

reasonable performance in recalling or predicting the sequence, the R-value was 0.550.

3.8.7.2 EO-MTRNN: Generalization Ability of Autonomous Hyperparameter Estimation

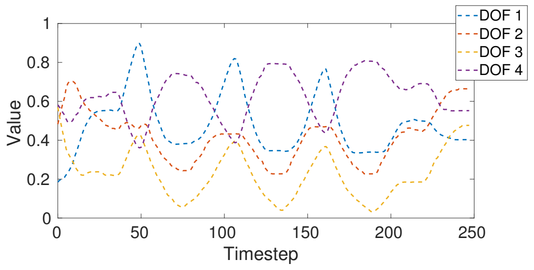
Here, the objective is to investigate the generalization ability of the proposed EO-MTRNN. The teaching data originated again from [230].

I evaluated whether the hyperparameters that were obtained based on particular teaching data would still enable the EO-MTRNN to learn different data. For this purpose, the longest sequence was selected that was taught in the experiment in [230]. The sequence encodes robot actions similar to but *not* the same as the one depicted in Figure 19. For example, the box is moved left to right, instead of up and down. This sequence was used as teaching data for the AHE. Then, the automatically estimated hyperparameters were used to train the network with the former behaviour sequence consisting of reaching, grasping, and moving up and down (Figure 19 and left side of Figure 20).

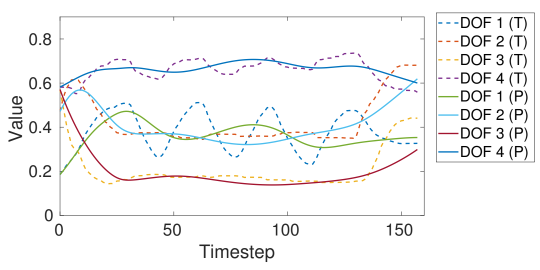
In sum, the obtained hyperparameters were used to learn a *different* sensory-motor sequence, i.e. not encountered during the AHE process. Teaching data and results of recall are shown in Figure 21. The preprocessing weight vector was the same 11-dimensional vector than in Section 3.8.6.2. The connective weights between sensory IO neurons and motor IO neurons were set to zero.

The AHE process yielded $N_{FC} = 30$, $N_{SC} = 18$, $\tau_{IO} = 64$, $\tau_{FC} = 57$, $\tau_{SC} = 290$. For training the different sequence (not seen during AHE process), the target MSE was set to $9.0 \cdot 10^{-6}$ that was reached after 10^6 epochs. The achieved R-value was 0.622.

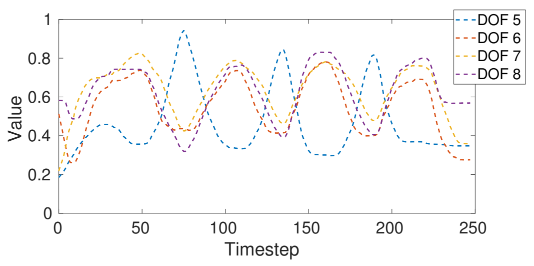
In this section and in Section 3.8.7.1, the values for the learning rates and momentum were the same as in Table 11.



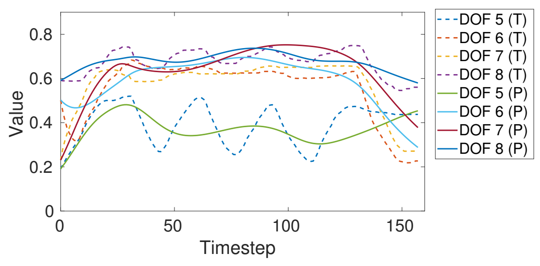
(a) Teaching data for AHE (DOF 1 to 4)



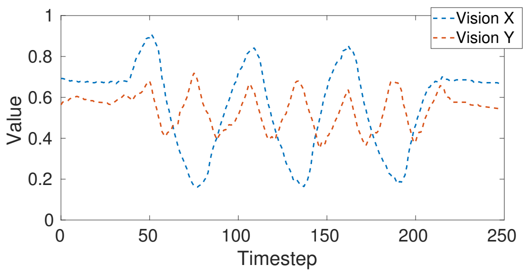
(b) Different teaching data (DOF 1 to 4, dashed lines) and its mental simulation (solid lines)



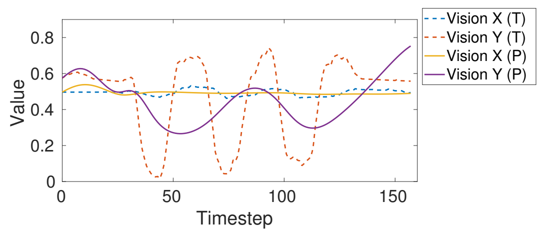
(c) Teaching data for AHE (DOF 5 to 8)



(d) Different teaching data (DOF 5 to 8, dashed lines) and its mental simulation (solid lines)



(e) Teaching data for AHE (2D vision)



(f) Different teaching data (2D vision, dashed lines) and its mental simulation (solid lines)

Figure 21 Generalization performance of the proposed EO-MTRNN at learning robot sensory-motor data. Left side (21(a), 21(c), 21(e)): Teaching data of the behaviour sequence used for AHE. The teaching data originated from [230]. The data encode reaching and grasping the box, and moving it left and right three times. Right side (21(b), 21(d), 21(f)): Dashed lines are the teaching data of the task shown to the robot in Figure 19. These data are different from the data on the left used for AHE. For example, the data on the left side encode moving the box left to right, instead of up and down. The solid lines are the mental simulation of the robot task by the EO-MTRNN. These results show *generalization ability*: The EO-MTRNN is able to sufficiently learn the task structure (reach & grasp the box, move it *up & down* three times), although it estimated its hyperparameters for another task (reach & grasp the box, move it *left & right* three times). The approximation can be further improved by increasing the number of generations ($N_{Gen.} > 10$) or by increasing the number epochs per individual of the network population of the AHE.

3.9. Discussion

3.9.1. Configurations of the Network

In the Sections 3.8.2, 3.8.3, 3.8.4 presenting the evaluation of the MTRNN block (i.e. without the evolutionary optimizer), I chose a hyperparameterization (Section 3.7.3) similar to the descriptions in [230], with a careful focus on the ratio between the timescales τ_{FC} and τ_{SC}.
The aim of the proposed benchmark training dataset is to capture a wide variety of possible sequences that a MTRNN could possibly encounter. The step-wise increment of the spatial dimensions of the benchmark sequences, e.g. from 1 to 2 to 4 etc., along with the different temporal lengths, should help to investigate how the learning ability of the network scales with the increase of spatial and temporal dimensions. Gaussian noise was added to the training data in order to simulate real application scenarios, for example when the network is supposed to learn sensory-motor data collected on a robot.
The evaluation of the learning ability compares four different network configuration modes, that are offered by the features preprocessing (*PP*) and early stopping (*ES*).
For the learning of one-dimensional sequences, Table 12 shows that a preprocessing of the input dimension increases the learning capability. A combination of activated preprocessing and early stopping (i.e. preprocessing *on* and early stopping *on*) has the best outcome.
For the learning of multi-dimensional sequences, Figure 10 shows that the combination of an active preprocessing and de-activated early stopping (preprocessing *on* and early stopping *off*) shows the best results. In addition, it was observed that preprocessing speeds up the BPTT convergence up to ten times, compared to configurations without preprocessing.
I used sigmoid activation for all units of the networks. Using other activation functions may influence the results, but that is beyond the scope of this work.

3.9.2. Optimization Performance

The implementation of SA-DE was validated by using the most important functions of the benchmark function table given in [28]. The obtained results concurred with [28] and show the ability of SA-DE to escape from a high number of local optima and to locate the global optimum.
The results in Figure 13 and Figure 14 demonstrate that the proposed optimization system can improve the learning capability of the MTRNN. The average improvement of 43 % for the training of single sequences was computed from the different evaluation cases shown in Figure 14. The particular network configuration with active preprocessing and early stopping (i.e. preprocessing *on* and early stopping *on*) was optimized, since this configuration yielded some weaknesses in learning multidimensional sequences with a length greater than 100 (see Figure 10(d)).
For the training of multiple sequences simultaneously, the evolutionary optimization of hyperparameters turned out to be beneficial particularly for learning sequences with increasing length, e.g. $L \geq 100$. An adverse (manual) choice of hyperparameters significantly deteriorates the learning performance, as was seen in the default cases in Figure 15. There, evolutionary optimization yielded a performance gain of up to 131 %.

All experiments were conducted using a conventional CPU that was an i7. Due to these computational limitations, I used a minimal optimization setting by $NP = 4$ and $N_{Gen.} = 10$, compared to the settings in many benchmark problems that are $NP = 100$ and $N_{Gen.} \geq 100$, like in [28]. Since the optimization run on CPU, this minimal setting with 4 individuals and 10 generations, including the retraining of the network, took several hours or even days to complete. Nevertheless, the optimization setting $NP = 4$ and $N_{Gen.} = 10$ already improved the learning capability for the given network configuration.
In most of the conducted experiments, the proposed EO-MTRNN delivered similar ratios of τ_{FC} and τ_{SC} as suggested in the literature [230]. It also showed that the number of context neurons has less impact on the learning performance than the timescales have.
Note that the SA-DE should only optimize the hyperparameters of the MTRNN, it should *not* replace the BPTT training algorithm including early stopping. A performance comparison of the BPTT against a SA-DE that optimizes all network parameters and internal representations (i.e. hyperparameters *and* connective weights as well as initial context potentials) is out of scope.

3.9.3. Application to Robot Data

Section 3.8.7 shows the performance of the proposed EO-MTRNN on data that were collected on a humanoid robot in [230].
I validated the training and recall of an action sequence by the EO-MTRNN, summarized in Figure 20. A very good sequence recall in [230] was compared to a recall where the AHE was switched off.
A sequence recall can be characterized as very good if the network recalls a sequence that very closely resembles the taught sequence, in other words, if the taught sequence can be exactly reproduced. Quantitatively, for very good recalls, the R-value is in the range of $0.9 < R < 1.0$. However, this may also include an overfitting, especially for $0.95 < R < 1.0$. In the range of $0.5 < R < 0.7$, the recall can be characterized as reasonable, since the taught task structure is still preserved, i.e. the basic actions such as reaching and grasping the box, and moving it in particular directions could be reproduced. Here, the difference to a very good recall is that the manipulated object is likely to be moved two times up and down instead of three times. This was observed in Figure 20, with an achieved recall of $R = 0.550$. This recall performance of the EO-MTRNN was reasonable, although I deliberately chose a non-optimal ratio of the timescales ($\tau_{IO} = 10$, $\tau_{FC} = 20$, $\tau_{SC} = 40$) and used relative few context neurons ($N_{FC} = 15$, $N_{SC} = 5$), compared to $N_{FC} = 60$ and $N_{SC} = 20$ in [230]. This showed that even in a *worst-case* scenario of network settings, where the AHE was switched off and the hyperparameters are adverse, the proposed network can still learn the task structure.
Then, I validated the generalization ability of the EO-MTRNN, i.e. how well the network can learn and recall data that were *not* part of the hyperparameter optimization process, see the results in Figure 21. For a good generalization ability, the hyperparameter optimization should be based on a rich variety of data. Good generalization refers to the ability of the network to recall data that the network was not optimized for through AHE. The quality of such recall is expected to be in the range of $0.5 < R \leq 0.8$. A high quality of recall (i.e. $R > 0.8$) cannot be

expected, since the network hyperparameters are not optimized for the data to be learned. For the optimization, a training sequence was chosen that contains oscillatory patterns covering a wide value range. The evolutionary optimizer computed $N_{FC} = 30$, $N_{SC} = 18$, $\tau_{IO} = 64$, $\tau_{FC} = 57$, $\tau_{SC} = 290$. Although τ_{IO} is slightly greater than τ_{FC}, their ratio is approximately one. The ratio between τ_{FC} and τ_{SC} is approximately 1 to 5, which concurs with the optimal ratio between τ_{FC} and τ_{SC} in [230]. The optimization setting was the same as before (i.e. $NP = 4$ and $N_{Gen.} = 10$). The resulting quality of recall was $R = 0.622$. Further improvement in approximating the teaching data (serving as ground truth) would be achieved by running the optimizer for more than 10 generations, yielding even better hyperparameters. Moreover, additional improvement in approximation might be obtained by further reducing the MSE.
Note that an investigation of the training and optimization of the entire dataset containing multiple sequences was out of scope due to computational limitations, i.e. CPU usage only. Network optimization would be significantly speeded up through GPU usage.
Besides the conducted simultaneous training of multiple benchmark sequences (in Section 3.8.6.2), a simultaneous training of multiple *robotic* sequences was not conducted due to time restrictions. As explained in Section 3.8.6.2, the study of simultaneous multiple sequence training took several days on a conventional computer with CPU (three cases of different length shown in Figure 15, evolutionary optimization for the third case of 1050 sample vectors took 35 hours).
I can conclude that the proposed EO-MTRNN can only scale up its performance to larger sets of data, e.g. on robots, if the algorithm is implemented on parallel processing hardware such as GPU. This means that any limitation of performance is due to the hardware the proposed algorithm runs on.

3.10. Summary

In this chapter, I extended a spatiotemporal learner (i.e. a modified MTRNN) by an evolutionary optimizer for AHE. This resulted in a proposed new learner (i.e. EO-MTRNN) that is able to improve itself over time in order to better capture the teaching data.
The benefit is that a cumbersome tuning of important network hyperparameters by a human expert can be avoided. A possible application of this self-improving spatiotemporal learner is its deployment within an agent or robot that can relearn over time if it needs to adapt to newly sampled teaching data, without any human intervention.
In order to validate the proposed spatiotemporal learner, I created a benchmark dataset and evaluated its performance starting with a minimal configuration containing only a few input-output dimensions and data samples. Then, the spatial and temporal dimension was each increased step by step during evaluation.
The introduction of an analytical pre- and postprocessing scheme has the benefit that this proposed version of the MTRNN does not require training of additional neural networks (e.g. topology preserving maps) pre- and postprocessing input-output data.

An early stopping method was also included as part of the training process. Results showed that a combination of active preprocessing and de-activated early stopping provides the best results for sequence learning.
I showed that an autonomous estimation of MTRNN hyperparameters by the evolutionary optimizer increased the learning performance. For a mixture of different benchmark sequences, ranging from 4 to 10 spatial dimensions and 50 to 150 temporal dimensions, the EO-MTRNN improved the learning performance compared to a non-optimized version by approximately 43 % in average without overfitting the given teaching data. The study of training multiple benchmark sequences simultaneously confirmed this performance gain and showed that the learned sequences can be clearly separated in the context memory of the network.
Moreover, I applied the proposed EO-MTRNN to sensory-motor data from a humanoid robot. The EO-MTRNN also regenerated data that were not part of the training set for the hyperparameter optimization process.

4. Chapter

Predictive Action Selector

In the previous chapter, I have proposed and validated a new type of spatiotemporal learner — the EO-MTRNN — that is based on the predictive coding principle. The EO-MTRNN can learn and predict sequences of sensory-motor data. Furthermore, its evolutionary optimization method allows it to find an ideal configuration in terms of hyperparameters.
In this chapter, the proposed EO-MTRNN will be part of a new action selection system: the PAS. In order to address the problem of poverty of stimulus [192, p. 260], [193], I propose the PAS as a new system for the generation of coordinated limb movements on various robot platforms. Thus, the development and validation of the PAS is the basis for a new cognitive architecture explained in the next chapter.
The experimental validation of the PAS was done on two different robots: NAO robot (version 5, body type H25) [173] and TOMM robot [52].
This chapter is structured as follows: Section 4.1 justifies why the new action selector is required and outlines the scientific goals. Section 4.2 carves up the problem of poverty of stimulus, it focuses on those sensory-motor stimuli that are needed for an (artificial) brain to generate limb movements in a coordinated manner. The poverty of stimulus constrains the sensory system that is modelled by the sensory-motor interface of the PAS in Section 4.3. Based on this sensory system, Section 4.4 describes the constrained degree of freedom exploration that I propose as a method for the autonomous generation of sensory-motor data required for the PAS. I evaluated the proposed constrained degree of freedom exploration on the robots NAO and TOMM. An important property of my proposed action selection method is that it generalizes from a minimum amount of sensory-motor data to meaningful behaviour. This generalization problem is shortly described in Section 4.5, deducing requirements for the method development. Section 4.6 outlines the components of the proposed PAS. The PAS itself consists of two spatiotemporal predictor blocks, each realized by the proposed EO-MTRNN. These two predictor blocks are outlined in Sections 4.6.2 and 4.6.3. Section 4.6.2 describes the self-motion predictor (SMP) that learns to predict the ego-motion of the robot, i.e. the self-generated motion of robot limbs seen from the robot's perspective. Section 4.6.3 describes the feature predictor (FP) that learns to predict the motion of external entities, e.g. external objects like a cup, that the robot interacts with. Section 4.6.4 explains how the SMP and FP are linked together by a method inspired by the action selection in the neocortex. Section 4.7 outlines two different modes of the proposed method of action selection: a first-stage prediction mode and a second-stage prediction mode. Section 4.8 describes experiments, in which two PAS modules learned to control the NAO robot (one PAS learned to control the head and the other one learned to control the arm). Sections 4.9 and 4.10 show the corresponding results. Section 4.11 describes the experiments, in which the PAS learned to control the TOMM robot to perform a multi-staged reaching. Section 4.12

shows the corresponding results. Section 4.13 thoroughly discusses the obtained results with reference to the scientific goals outlined at the beginning of this chapter. Finally, Section 4.14 summarizes this chapter and bridges to the next.
This chapter contains many algorithmic descriptions. The typographical convention is the following: Data types and functions are italicized, whereas parameters and variables are in normal font unless they are expressed as mathematical symbols such as scalars (italic) or vectors (bold). Computational steps are enumerated. Important steps are described by comments within curly brackets. Whenever an OpenCV function is used for visual processing, it is indicated in the comments by "OCV" and also mentioned in the algorithm caption.
Note that I published parts of this chapter in [220], [221], [34], and [222].

4.1. Motivation for a New Type of Action Selector

The PAS overcomes the limitation of the action selection mechanisms detailed in Chapter 2, in particular in Table 2 and in Table 5.
The PAS is aimed at realizing four different goals, outlined in Table 16. These goals are

Priority	**Goals of PAS**	**Limitations (of existing methods) overcome**
1	Learning of hand-eye coordination from minimum amount of samples, without human intervention	Predictive coding-based methods [192] rely on a human teaching every action; they do not support autonomous acquisition of hand-eye coordination.
2	Learning the dynamics of external entities like objects by observing them	Theoretical work about predictive coding often provides the dynamics beforehand, such as in [60, p. 135].
3	Robustness to temporal loss of sensory data	Traditional methods, e.g. [83], [159], stop execution if sensory data are lost
4	Integration of action selection and action generation into one framework	Motive-driven action selection [209], [211], [206], [207] de-couples action selection from action generation, at the expense of adaptivity and flexibility

Table 16 Goals realized by PAS and the corresponding limitations that are overcome. The goals are sorted according to priority, with the highest priority (1) starting on top.

ordered according to priority, starting with the most important goal on the top. The first two goals are logical extensions of existing predictive coding frameworks. While the EO-MTRNN facilitates robustness to temporary loss of data by its prediction ability, it requires teaching data. The teaching data should be generated by the robot and the new action selection method should make sense of very little data to bootstrap coordinated motions. The acquired data are learned by the EO-MTRNN. However, some low-level goal information has to be encoded in the method in order to direct the learned sequences to it. The goal information is in the sensory domain; when the goal information is combined with the sequence prediction, the new selection method yields coordinated motion, i.e. motions that are directed to reach the goal.

4.2. Poverty of Stimulus — Sensory-Motor Samples for the First Robot Behaviour

An agent's knowledge and skills result from complex interactions with the environment and other agents. When tracing the interaction history to the beginning, developmental agents[1] face the same problem: poverty of stimulus [192, p. 260], [193], originally introduced by Chomsky [45] in the domain of language acquisition.
The problem of poverty of stimulus deals with the fact that developmental agents have access to only a few amount of sensory patterns. This is mainly due to their underdeveloped sensory and motor system [197], [109], [211]. At this point, deep learning approaches [161] cannot be directly applied, since they require a considerable amount of data that does not exist in this early stage due to morphological and other physical sensory-motor constraints [139].

4.2.1. Questions Deduced from the Poverty of Stimulus

For a developmental robot, this poverty of stimulus leads to following questions:

- What are the constraints to sensors and motors? In particular, what is perceived, and what joints are able to move and how?
- In relation to the first question, what kind of (sensory-motor) samples should be collected?
- How can a new action selection method exploit and generalize from the limited set of samples?
- Does the sensory system develop, or in other words, is a developmental modelling of the sensory system considered?

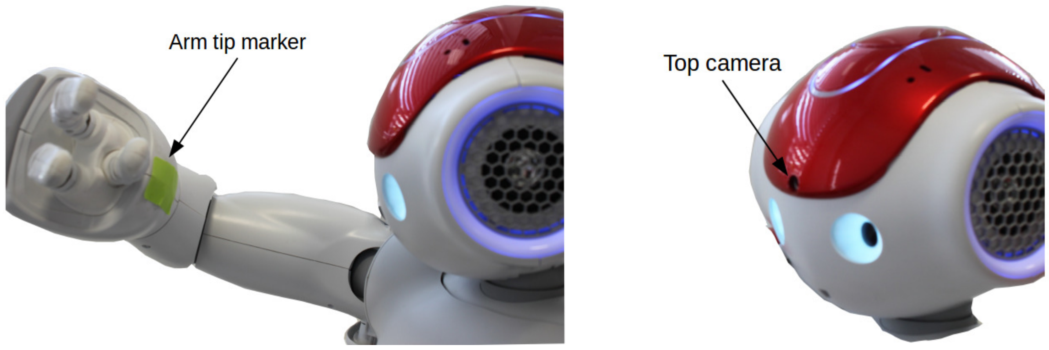
Since the poverty of stimulus has a wide range of implications, there may be more questions that are out of scope. Also, the development of the sensory system itself is not considered in this research.
In this chapter, the focus is set by the first three itemized questions. They are the basis for the construction of the PAS.

4.2.2. Morphological and Perceptual Constraint

The robot has to have the means to visually recognize parts of its own arm that it learns to control. A training of classifiers (e.g. deep learning networks) for marker-less object recog-

[1] These can be infants in their first weeks after birth as well as developmental robots.

nition was avoided, since the objective here is to model the poverty of stimulus which has to cope with very little data, i.e. with poor visual features.
Thus, the sensory system is constrained to the perception of basic colour that is red, green, and blue. As such, a green marker was attached to the inner surface of the NAO robot's arm tip or wrist, see Figure 22.



(a) Right arm with green marker

(b) Head with top camera used for visual input

Figure 22 NAO robot with marker attached to the end of its arm. Side view (22(a)) of the NAO robot (right arm and head). A green marker was attached to the inner surface of the robot's arm tip or wrist. NAO's top camera was used (22(b)).

4.3. The Sensory-Motor Interface of the PAS

This section explains the sensory-motor interface of the PAS. The interface is shown in Figure 23. The interface consists of the following components:

- Visual feature cells
- Visual feature extractor
- Joint angle normalizer
- Joint angle de-normalizer

The main sensor modality is vision. Optionally, a simple tactile feedback is also used that is a short tactile touch event (i.e. a tap) sending a binary event signal (i.e. *touched*) to the PAS. This tactile feedback can be in principle replaced by a single key stroke on a computer keyboard, but it is more practical if the robot platform supports it.
The vision part of the interface was implemented by using OpenCV [130] functions provided by the robot operating system (ROS) [129]. The ROS version used was *Kinetic Kame*.

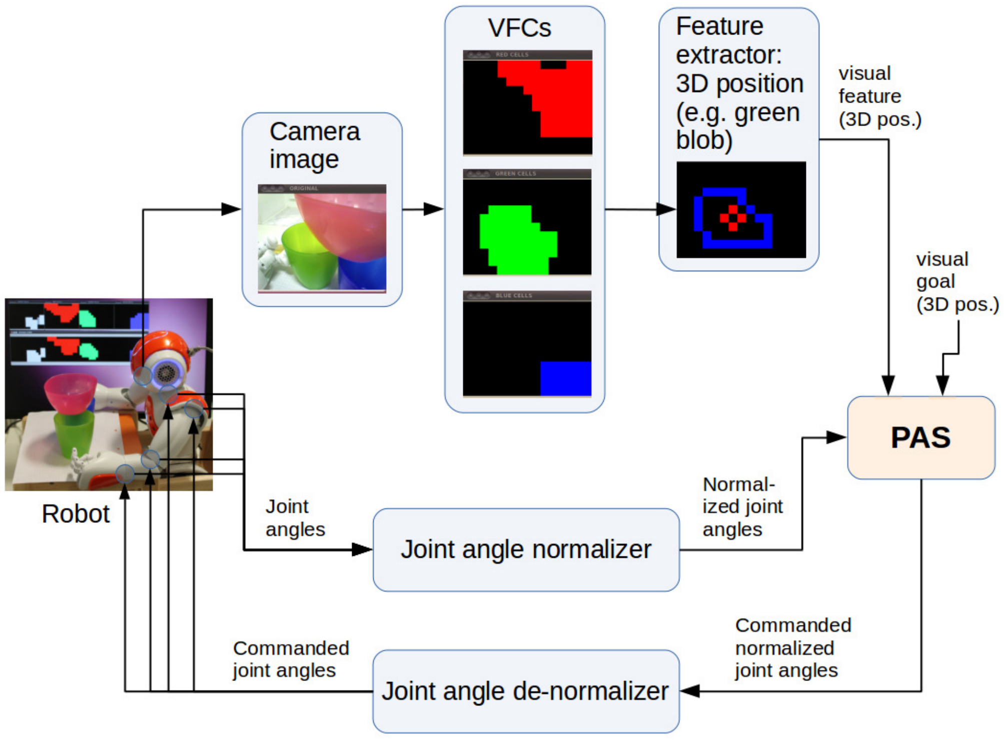
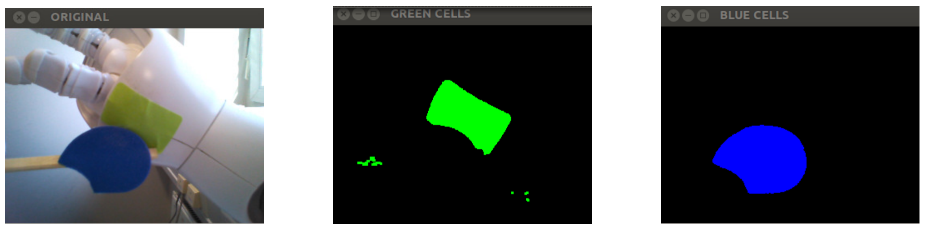


Figure 23 Sensory-motor interface of the PAS. The top part shows the processing of visual input that is a 3D position of an environmental feature, e.g. an object of interest, in the robot's camera. The bottom part shows the processing of proprioceptive input and output in terms of joint positions. Note that the state of the visual feature cells (VFCs) can be optionally fed into a HHOP encoding a type of episodic memory.

4.3.1. Visual Feature Cells

The VFCs model a simple artificial retina. It extracts simple visual features from a raw image of a camera. The visual features are red, green, and blue colour. An optional feature is the shape of a foreground object projected on the image plane. The image resolution is scalable. Figure 24 shows an example. For each colour to be extracted, the procedure is the same: The



(a) Original camera image (b) Green cells state (c) Blue cells state

Figure 24 Example scenario of VFCs output. The original camera image 24(a) shows the robot's current FOV, with its right arm reaching a target object (blue marker on chopstick). The corresponding state of the green cells 24(b) and blue cells 24(c) represent the arm tip and the object, respectively. The image resolution was 640×480 pixels in the camera image as well as in the images of the red, green, and blue cells. The state of the red cells is not shown, since a vast majority of them is inactive due to the given scenario on the left.

first step converts the raw colour images from the camera into hue-saturation-value (HSV) image, since the HSV colour space is more robust against changes of environmental light compared to red-green-blue (RGB) colour space. Then, the image is scaled to a fixed size

with a nearest neighbour interpolation. In the next step, a threshold function filters out the particular colour and dilation is applied. The result is a binary image that represents the 2D shape of those parts of a foreground object that contain the particular colour.
Algorithm 2 outlines the detailed working principle. Prior to the execution of Algorithm 2, the incoming camera image is transformed from the ROS image format to an OpenCV image format (BGR8).

Algorithm 2 Visual feature cells (VFCs). Comments are in curly brackets. The usage of an OpenCV function is indicated by "OCV" in the comment.

```
Input: Mat src_img   {source image, BGR8-format of OpenCV}
Output: Mat cells_ipl   {target image in which each pixel represents a binary feature cell sensitive to
                         a particular colour}
Parameters: int $s$, int tuple (min1, min2, min3), int tuple (max1, max2, max3)   {scale factor, tuples
            of minimum and maximum values in HSV-space, respectively}
Variables: Mat hsv_img, IplImage hsv_img_ipl, IplImage scaled_hsv_img_ipl, Mat img_thresh_colour

1: cvtColor(src_img, hsv_img, CV_BGR2HSV, 3)   {OCV: Convert from raw colour image into HSV
                                                image (3 channels)}
2: hsv_img_ipl $\leftarrow$ hsv_img   {Store in Ipl-format}
3: {OCV: Scale the image to a fixed size using 8 bit depth and 3 channels for the target image}
   scaled_hsv_img_ipl $\leftarrow$ cvCreateImage(cvSize(hsv_img.cols/$s$, hsv_img.rows/$s$), 8, 3)
4: {OCV: Scale the HSV image with nearest neighbour interpolation}
   cvResize(&hsv_img_ipl, scaled_hsv_img_ipl, CV_INTER_NN)
5: {OCV: Create target image containing the blobs of particular colour, 8 bit, 1 channel}
   img_thresh_colour $\leftarrow$ cvCreateImage(cvSize(hsv_img.cols/$s$, hsv_img.rows/$s$), 8, 1))
6: {OCV: Threshold depending on the required colour encoded by the range from (min1, min2, min3)
   to (max1, max2, max3)}
   cvInRange(scaled_hsv_img_ipl, cvScalar(min1, min2, min3), cvScalar(max1, max2, max3),
             img_thresh_colour)
7: {OCV: Create target image that emulates the feature cells sensitive to the particular colour}
   cells_ipl = cvCreateImage(cvSize(hsv_img.cols/$s$, hsv_img.rows/$s$), 8, 1)
8: {OCV: Dilate the thresholded image by using a 3-by-3 kernel with centred anchor and 1 iteration,
   and save it into cells_ipl}
   cvDilate(img_thresh_colour, cells_ipl, NULL, 1)
9: return cells_ipl
```

Note that the scale factor was $s = 1$, i.e. the image resolution was not changed. The default image resolution was 640×480 pixels. If $s > 1$, the image will be reduced in its resolution. The setting $s > 1$ was used when the visual pattern delivered by this algorithm was fed into a HHOP [219] that encodes a type of episodic memory[2].
The colour sensitivity of the VFCs is determined by the tuples (min1, min2, min3) and (max1, max2, max3). Three types are implemented, yielding three different types of calls of Algorithm 2. The type is determined by the following parameters:

- Red: min $= (160, 100, 50)$ and max $= (200, 255, 255)$
- Green: min $= (30, 100, 50)$ and max $= (50, 255, 255)$

[2] For the HHOP, a smaller resolution is required due to computational limitations regarding the size of the random access memory. The HHOP is *not* part of the PAS. The HHOP memorizes the visual patterns delivered by the VFCs. In potential future work, the HHOP can be used as additional trigger mechanism to initialize or switch between actions.

- Blue: min $= (100, 150, 50)$ and max $= (120, 255, 255)$

Thus, for every incoming camera image, the three different calls of Algorithm 2 are executed in parallel[3], yielding a synchronized activity pattern of red, green, and blue cells.

4.3.2. Visual Feature Extractor

Each time the VFCs are updated and synchronized, they deliver colour blobs as input to the visual feature extractor. The visual feature extractor computes the three-dimensional position of the colour blobs relative to the camera image plane. In particular, two different types of colour blobs are of interest: the colour blob representing an external object of interest which the robot can interact with, and the colour blob representing the robot's arm tip (Section 4.2.2). Algorithm 3 shows the procedure implementing the position feature extraction. Note that the sensory system of the PAS does not require stereo vision. Indeed, stereo vision offers the benefit of distance estimation by using triangulation. However, not every robot supports stereo vision, for example the NAO. In order to address a wide variety of robot platforms, the sensory system of the PAS should be able to cope with object distances with a monocular camera. For this reason, the position computation estimates the object distance based on the size of the colour blob.
Algorithm 4 describes the computation of the 3D position of the colour blob that represents either the robot's arm tip or the external object for interaction. Algorithm 4 contains a function *normalizePosAndEstimateDist()* that is described by Algorithm 5 in order to estimate the distance and to normalize the position.
The end result is a normalized 3D position $\mathbf{v}_f$ of the visual feature. Mathematically, it has the form $\mathbf{v}_f = (v_x \quad v_y \quad v_z)^T$. Each element is in the interval

$$(0.0, 1.0) := \{x \in \mathbb{R} \mid 0.0 < x < 1.0\} \tag{38}$$

with x representing the vector element. The normalization step is necessary, since the PAS uses the EO-MTRNN for the processing of visual features and the network operates with values of the interval (38).

4.3.3. Joint Angle Normalizer

The joint angle normalizer is pre-processing the motor data from the robot. Here, the motor data are represented by the angular position of joints as the only physical quantity. As shown in Figure 23, the joint angles are normalized before they are fed into the PAS. In this case, angle normalization means that each DOF angle of a joint is mapped to values between 0.0 and 1.0, which is the same interval (38) described in Section 4.3.2. In the following, the joint angles are referred to as proprioceptive data. Similar to the normalization of 3D position of visual features, the normalization of joint angles is necessary as well, since the proprioceptive data are processed by the EO-MTRNN of the PAS algorithm. A value of a particular DOF is

[3] In ROS, this is achieved via message callback functions that are synchronized for every incoming message. It is quasi-parallel.

Algorithm 3 Visual feature extractor. Comments are in curly brackets. The usage of an OpenCV function is indicated by “OCV” in the comment.

```
Input:
const ImageConstPtr& red_cells, green_cells, blue_cells   {the VFCs output as time-synchronized
                                                           ROS images}
Output: double v_obj_pos[3], v_arm_tip_pos[3]   {visual features: normalized 3D positions of the
                                                 object and the arm tip, respectively, which can be
                                                 published via ROS messaging}
Parameters: int blob_reliab_thresh_obj   {minimum contour size of the blob representing the object},
            int blob_reliab_thresh_arm_tip   {minimum contour size of the blob representing
                                              the arm tip},
            char feature   {value representing the position computation of either the external object
                            (‘o’) or the robot’s arm tip (‘a’)},
            int hp_contour_obj   {value of the blob contour representing the object at home position
                                  distance relative to the robot’s camera},
            int hp_contour_arm_tip   {value of the blob contour representing the arm tip at home
                                      position distance relative to the robot’s camera}
Variables:
cv_bridge::CvImagePtr red_cells_ptr, green_cells_ptr, blue_cells_ptr   {image pointers for the
                                                                        conversion of ROS image
                                                                        message to OpenCV format},
IplImage red_cells_ipl, green_cells_ipl, blue_cells_ipl   {the state of the different feature cells
                                                           encoded in Ipl-format},
Point3d blob_pos   {temporary storage of the normalized 3D position}

    {OCV: Transform the incoming ROS images into OpenCV images (lines 1 to 3)}
 1: red_cells_ptr ← cv_bridge::toCvCopy(red_cells, enc::BGR8)
 2: green_cells_ptr ← cv_bridge::toCvCopy(green_cells, enc::BGR8)
 3: blue_cells_ptr ← cv_bridge::toCvCopy(blue_cells, enc::BGR8)
    {Store the image in Ipl-format (lines 4 to 6)}
 4: red_cells_ipl ← red_cells_ptr->image
 5: green_cells_ipl ← green_cells_ptr->image
 6: blue_cells_ipl ← blue_cells_ptr->image
    {Compute the position of the external object encoded by the blue cells (line 7)}
 7: blob_pos ← computeBlobPosition(&blue_cells_ipl, blob_reliab_thresh_obj, ‘o’, hp_contour_obj)
    {Store the object position for ROS messaging (lines 8 to 10)}
 8: v_obj_pos[0] ← blob_pos.x
 9: v_obj_pos[1] ← blob_pos.y
10: v_obj_pos[2] ← blob_pos.z
    {Compute the position of the right arm tip encoded by the green cells (line 11)}
11: blob_pos ← computeBlobPosition(&green_cells_ipl, blob_reliab_thresh_arm_tip, ‘a’, hp_con-
    tour_arm_tip)
    {Store the arm tip position for ROS messaging (lines 12 to 14)}
12: v_arm_tip_pos[0] ← blob_pos.x
13: v_arm_tip_pos[1] ← blob_pos.y
14: v_arm_tip_pos[2] ← blob_pos.z
15: return
```

Algorithm 4 Function *computeBlobPosition()* for computing the normalized 3D position of a target blob that represents either the arm tip or the external object. Comments are in curly brackets. The usage of an OpenCV function is indicated by "OCV" in the comment.

Full name:
*computeBlobPosition(IplImage** cells_ipl*, int* blob_reliab_thresh*, char* feature*, int* hp_contour*)*
Input: *IplImage** cells_ipl {pointer on the input image (delivered by Algorithm 2)},
int blob_reliab_thresh, *char* feature, *int* hp_contour {parameters of Algorithm 3}
Output: *Point3d* result {normalized position (3D) of the target blob relative to the image plane of monocular camera}
Parameters: *double* hp_distance_offset_obj, *double* hp_distance_offset_arm_tip {value to bias the normalized distance of the external object and the arm tip, respectively}
Variables: *Mat* img, *vector< vector<Point> >* contours, *vector<Vec4i>* hierarchy, *double* contour_size, *double* max_mom, *int* max_mom_idx, *vector<Moments>* mom(contours.*size()*), *Point2d* blob_pos

```
 1: cells ← cvarrToMat(cells_ipl).clone()   {OCV: Transform Ipl image to Mat image and copy it}
 2: cvtColor(cells, img, CV_RGB2GRAY, 1)   {OCV: Convert to single channel image}
    {OCV: Find contours (line 3)}
 3: findContours(img, contours, hierarchy, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE)
    {Return a default value ($-1$) in case of no contours at all (lines 4 to 9)}
 4: if contours.size() $== 0$ then
 5:    result.x ← $-1.0$
 6:    result.y ← $-1.0$
 7:    result.z ← $-1.0$
 8:    return result
 9: end if
    {Get the moments of each contour (lines 10 to 12)}
10: for $i \leftarrow 0; i <$ contours.size(); $i \leftarrow i + 1$ do
11:    mom[$i$] ← moments(Mat(contours[$i$])), false)   {OCV: Get the moments}
12: end for
    {Get the index of the largest contour (lines 13 to 20)}
13: max_mom ← mom[$0$].m00
14: max_mom_idx ← $0$
15: for $i \leftarrow 1; i <$ contours.size(); $i \leftarrow i + 1$ do
16:    if mom[$i$].m00 $>$ max_mom then
17:       max_mom ← mom[$i$].m00
18:       max_mom_idx ← $i$
19:    end if
20: end for
    {Check the reliability of the target blob by using the given minimum contour size (lines 21 to 26)}
21: if max_mom $<$ blob_reliab_thresh then
22:    blob_pos ← Point2d($-1.0$, $-1.0$)   {target blob is not reliable}
23: else
    {Get the mass centre of the largest blob, i.e. 2D position in pixels (line 24)}
24:    blob_pos ← Point2d(mom[max_mom_idx].m10/mom[max_mom_idx].m00, mom[max_mom_-
    idx].m01/mom[max_mom_idx].m00)
25:    contour_size ← max_mom   {Update the contour size}
26: end if
    {Check if the target blob is visible, estimate its distance, and compute its normalized position
    (line 27)}
27: result ← normalizePosAndEstimateDist(blob_pos, cells_ipl->width, cells_ipl->height, hp_contour,
    contour_size, feature, hp_distance_offset_obj, hp_distance_offset_arm_tip)
28: return result
```

Algorithm 5 Function *normalizePosAndEstimateDist()* as part of Algorithm 4. Comments are in curly brackets.

Full name:
normalizePosAndEstimateDist(Point2d blob_pos*, int* width*, int* height*, int* hp_contour*,*
double contour_size*, char* feature*, double* hp_distance_offset_obj*,*
double hp_distance_offset_arm_tip*)*

Input: *Point2d* blob_pos {already contains the 2D position in pixels},
int width, height {image size in pixels},
int hp_contour {contour size of target blob when the corresponding physical entity (object or arm tip) is in home position distance},
double contour_size {contour size of current target blob},
char feature {see parameters of Algorithm 3},
double hp_distance_offset_obj, hp_distance_offset_arm_tip {see parameters of Algorithm 4}

Output: *Point3d* result

```
1:  if blob_pos.x $\geq 0.0$ and blob_pos.y $\geq 0.0$ then
    {Target blob is visible, thus compute the normalized position (2D) in the image plane (lines 2 to 3)}
2:    result.x $\leftarrow$ blob_pos.x / width
3:    result.y $\leftarrow$ blob_pos.y / height
    {Estimate the distance by using the contour size (lines 4 to 8)}
4:    if feature $==$ ‘a’ then
5:      result.z $\leftarrow$ hp_contour / contour_size - hp_distance_offset_arm_tip
6:    else if feature $==$ ‘o’ then
7:      result.z $\leftarrow$ hp_contour / contour_size - hp_distance_offset_obj
8:    end if {this yields either the distance of the arm tip ‘a’ or the object ‘o’}
    {Check the upper bound (lines 9 to 11)}
9:    if result.z $\geq 1.0$ then
10:     result.z $\leftarrow 0.999$
11:   end if
    {Check the lower bound} (lines 12 to 14)
12:   if result.z $\leq 0.0$ then
13:     result.z $\leftarrow 0.001$
14:   end if
15: else
    {Target blob is not visible, thus return the default value indicating that the feature is not present (lines 16 to 18)}
16:   result.x $\leftarrow -1.0$
17:   result.y $\leftarrow -1.0$
18:   result.z $\leftarrow -1.0$
19: end if
20: return result
```

normalized according to the linear normalization function

$$p = m \cdot \theta + c \tag{39}$$

where θ denotes the DOF angle in radians and p denotes the corresponding normalized DOF angle in the interval $(0.0, 1.0)$. The gradient m and the intersect c are computed as follows:

$$m = \frac{1}{\theta_u - \theta_l} \tag{40}$$

$$c = -\frac{\theta_l}{\theta_u - \theta_l} \tag{41}$$

with θ_l and θ_u denoting the lower and the upper angle limit of the DOF, respectively. These angles limits are in radians and have to be provided as part of the robot's technical specification. In addition, a lower and upper threshold check is done to ensure the interval $(0.0, 1.0)$. The minimum normalized DOF value is set to $p_l = 0.001$ if $p < 0.001$ and the maximum normalized DOF value is set to $p_u = 0.999$ if $p > 0.999$. These are the numerical bounds of the values fed into the EO-MTRNN, these bounds are the same for the computation of the 3D position $\mathbf{v}_f$ of the visual feature in Section 4.3.2.
Angle normalization requires that the lower and the upper limit of each DOF that is to be controlled by the PAS are known.

4.3.4. Joint Angle De-Normalizer

The joint angle de-normalizer is post-processing the motor data computed by the PAS. Here, the motor data are represented by the normalized target positions of the joints controlled by the PAS. The normalized target positions are de-normalized, i.e. mapped back to angular target positions, according to the linear de-normalization function

$$\theta^{(cmd)} = m \cdot p^{(cmd)} + c \tag{42}$$

where $\theta^{(cmd)}$ denotes the commanded DOF angle in radians and $p^{(cmd)}$ denotes the corresponding normalized DOF angle in the interval $(0.0, 1.0)$. The scalar $p^{(cmd)}$ is an element of the proprioceptive vector $\mathbf{p}^{(cmd)}$ computed by the PAS. Each vector element represents a target position of a particular DOF controlled by the PAS. For example, in case of a two-dimensional vector $\mathbf{p}^{(cmd)} = (p_1^{(cmd)} \quad p_2^{(cmd)})^T$, $p_1^{(cmd)}$ is the commanded position of DOF 1 and $p_2^{(cmd)}$ is the commanded position of DOF 2. In Equation (42), the gradient m and the intersect c are computed according to

$$m = \theta_u - \theta_l \tag{43}$$

$$c = \theta_l \tag{44}$$

with θ_l and θ_u denoting the lower and the upper angle limit of the DOF, respectively. These are the same limits as in case of the joint angle normalizer in Section 4.3.3. As additional safety measure, an overshooting of the angle limits is avoided by setting $\theta^{(cmd)} = \theta_l$ if $\theta^{(cmd)} < \theta_l$ and by setting $\theta^{(cmd)} = \theta_u$ if $\theta^{(cmd)} > \theta_u$.

4.4. Constrained Degree of Freedom Exploration for Generating Training Data for the PAS

I propose a constrained exploration of DOF that is a method to address goal (1) listed in Table 16. Modelling the poverty of stimulus (Section 4.2), the proposed method generates a minimum amount of sensory-motor samples, where each sample itself is primitive in the sense that it contains position, from both visual and motor space, as the only physical quantity.

4.4.1. Biological Inspiration and Benefit

While the method of DOF exploration already exists in related work, such as motor babbling[4] [96], [158], [188], [83] and goal babbling [148], [18], [147], [159], the proposed method here is characterized by its limited range of DOF motion, i.e. emphasizing *constrained* motion. Constrained DOF exploration moves each DOF back and forth only once, with one DOF moving at a time within a limited range, e.g. only 5 % of the total range. The benefit of this proposed method is a drastic reduction of the movement space and the number of training samples. This comes along with the poverty of stimulus: at the very beginning of mental development, the robot does not have the ability to coordinate its motion, it has to first learn a sensory-motor mapping. Although the data for acquiring that mapping are very limited, they can simplify the learning. Numerous studies, such as [23], [66], [109], [211], provide evidence that narrowing down the movement space is a property of the biological motor system in an early developmental stage. By narrowing down the range of motion, the training data are reduced and less time is spent to acquire an early sensory-motor mapping, which in turn can be refined later during the developmental process.
To further model the early developmental stage of the motor system, it is assumed that the agent has some kind of primitive mechanism enabling it to move its joints. This is biologically-plausible due the primary role of the central nervous system, which is the control of movements [226], although the motions are uncoordinated in this early stage.

4.4.2. Technical Realization

Technically, a proportional control (P control) models the earliest developmental stage and realizes the first movements of the joints. In order to learn the coordination of a particular limb, e.g. arm or head/neck, the P control moves each DOF of the limb[5] from its current angle position h (home position) into a target position $h + l$. Then, the P control moves the DOF back into home position, before it continues to move the DOF into the target position $h - l$ and then again back into home position. During the DOF motion from h to $h + l$ and during

[4] Motor babbling is also referred to as body babbling.
[5] Note that only one particular DOF of the limb is moving at a time while the others stay fixed.

the DOF motion from h to $h-l$, sensory-motor samples are collected in the form

$$\mathbf{s}_i^T = (\mathbf{v}_f^T \quad \mathbf{p}^T) \tag{45}$$

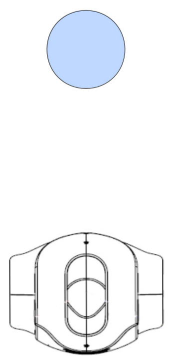
with $\mathbf{v}_f$ denoting the normalized 3D position of the visual feature (Section 4.3.2) and $\mathbf{p}$ denoting the normalized angle positions of all DOF of the limb to be controlled. A requirement here is that a particular visual feature of interest has to be visible in the robot's FOV. This requirement is necessary for learning the relation between perceptual (sensory) samples and motor samples, i.e. the relation learned by the PAS.
For the proposed exploration method, the visual input has to deliver one of two different types of features depending on the body part that the PAS should learn to control:

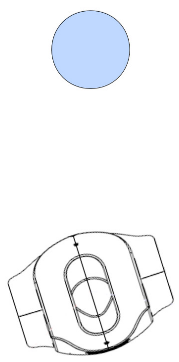
- External object (e.g. a cup, a box, etc.) that is used to learn the coordination of the head/eyes

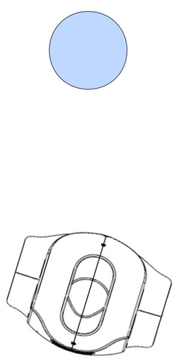
- Arm tip, hand, or end-effector that is used to learn the coordination of the arm

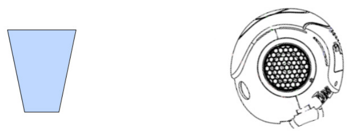
When learning to coordinate the head/neck joints, the visual feature of interest is the position of an external object (e.g. a cup) that is stationary during the exploration procedure. Figure 25 shows the principle of constrained DOF exploration when applying it to the 2 DOF head/neck joint of the NAO robot in order to learn to coordinate the head or the eyes[6] in a meaningful manner. An additional sequence is recorded when the limb or body part is in home position, this sequence is referred to as idle sequence (Figure 25(a)).
Note that this principle of exploration is the same for any limb that the robot should learn to control, independent of the number of DOF. Given n DOF of a particular limb to explore, the total number of sequences to record is $2n+1$ (up and down sequence per DOF plus the idle sequence). The idle sequence contains a fixed number of samples. In the implementation of this method, the idle sequence has 4 samples, no matter whether head or arm is explored.
When learning to coordinate the arm joints, the visual feature of interest is the position of the arm tip or hand. Due to the morphological and perceptual constraint (Section 4.2.2), a coloured marker was attached to the robot's wrist. Perception of that marker through the sensory interface of the PAS (Section 4.3) yields the required position of the arm tip or hand. The constrained DOF exploration then moves each DOF of the arm one after the other, starting from the shoulder joints and ending with the wrist joint[7]. Similar to the exploration of the head/neck DOF, it is important that the arm tip stays within the robot's FOV during the entire exploration process.
Figure 26 shows an excerpt of a constraint DOF exploration of the head as well as a constrained DOF exploration of the arm, both executed on the NAO robot. Algorithm 6 details the working principle. This algorithm is called through a ROS callback function at a particular frequency. Upon each call, it generates a motor command for each DOF that is to be explored.

[6] Note that the eyes in terms of cameras are fixed in the NAO robot's head, i.e. their orientation is determined by the 2 DOF head/neck joint.
[7] On the NAO robot, the fingers were neglected, since they are relatively small compared to the arm.


(a) Top view: Idle state


(b) Top view: Yaw up


(c) Top view: Yaw down


(d) Side view: Pitch up

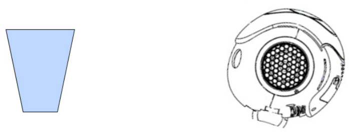
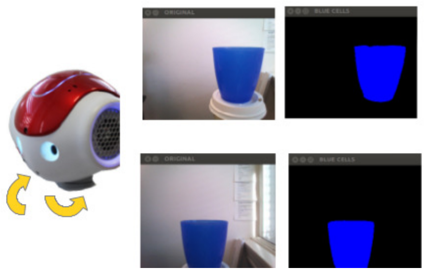

(e) Side view: Pitch down

Figure 25 Principle of constrained DOF exploration applied to the 2 DOF head/neck joint of the NAO robot. The object of interest (here, a blue cup) needs to be visible in the robot's FOV during the exploration. Every case depicted (i.e. 25(a) to 25(e)) represents a sensory-motor sequence that is recorded for a subsequent training of the PAS. Note that the cases "up" and "down" represent the angle position of each DOF actuated, and not the geometrical direction. Given the home position h for each DOF, "up" represents the final position $h + l$ and "down" represents the final position $h - l$. Besides the angular position of the actuated DOF, the 3D visual position of the object in the robot's FOV is also recorded.


(a) Exploring the head/neck DOF, visual feature is the 3D position of the blue cup in the robot's FOV

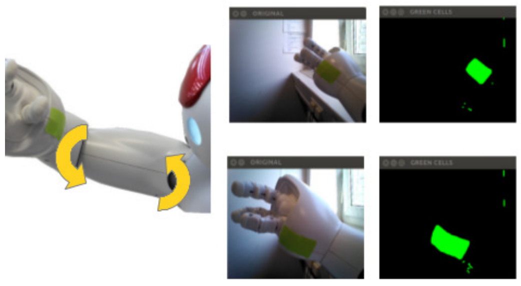

(b) Exploring the arm DOF, visual feature is the 3D position of the green arm tip marker in the robot's FOV

Figure 26 Excerpt of constrained DOF exploration of head (26(a)) and arm (26(b)) on the NAO robot. In each case, the 3D position in the robot's FOV is delivered by the sensory interface of the PAS, as explained in Section 4.3. The perceived sensory (i.e. visual) change that is caused by each DOF motion is of key importance for the subsequent acquisition of coordination skill by the PAS.

Algorithm 6 Function *exploreDOF()* implementing the constrained DOF exploration. Comments are in curly brackets.

Full name: *exploreDOF(double** motor_in*, double** motor_out*, double** hp*, int* num_dof*, int* limb*)*
Input: *double** motor_in {motor input, i.e. current normalized position of each DOF to be explored},
*double** motor_out {motor output, i.e. target normalized position of each DOF},
*double** hp {normalized home position of each DOF to be explored},
int num_dof {number of DOF to be explored},
int limb {body part for exploration, i.e. either head (0), left arm (1), or right arm (2)}
Output: *bool* result {indicating whether the exploration of the body part is finished or not}
Variables: *static int* ctr, dof_idx, *static bool* up, down, home_pos

```
    {Initialize variables (lines 1 to 6)}
 1: ctr ← $0$
 2: dof_idx ← $0$
 3: up ← $false$
 4: down ← $false$
 5: home_pos ← $false$
 6: result ← $false$
    {If all DOF have been explored, reset the variables and indicate the finish (lines 7 to 15)}
 7: if (dof_idx == num_dof) then
 8:    ctr ← $0$
 9:    dof_idx ← $0$
10:    up ← $false$
11:    down ← $false$
12:    home_pos ← $false$
13:    result ← $true$
14:    return result
15: end if
    {Record the idle sequence, its motor part contains the home position (lines 16 to 23)}
16: if ctr $== 0$ then
17:    int sample_idx ← $0$
18:    for $i \leftarrow 0; i < 4; i \leftarrow i + 1$ do
19:       recordSampleSMP(sample_idx, motor_in, num_dof, limb)
20:       sample_idx ← sample_idx $+ 1$
21:    end for
22:    recordSampleSMP($-1$, motor_in, num_dof, limb)    {Sequence end mark}
23: end if
    {Move DOF up and down as part of the exploration process (lines 24 to 42)}
24: if up $== false$ and down $== false$ then
25:    up ← moveDOFup(dof_idx, motor_in, motor_out, hp, num_dof, limb)
26: end if
27: if up $== true$ and home_pos $== false$ then
28:    home_pos ← moveDOFtoHP(dof_idx, motor_in, motor_out, hp)
29:    if down $== true$ and home_pos $== true$ then
30:       dof_idx ← dof_idx $+ 1$
31:       up ← $false$
32:       down ← $false$
33:       home_pos ← $false$
34:    end if
35: end if
36: if up $== true$ and home_pos $== true$ and down $== false$ then
37:    down ← moveDOFdown(dof_idx, motor_in, motor_out, hp, num_dof, limb)
38:    if down $== true$ then
39:       up ← $true$
40:       home_pos ← $false$
41:    end if
42: end if
43: ctr ← ctr $+ 1$    {Increase counter of recorded sequence}
44: return result
```

Algorithm 6 contains the sub-function *recordSampleSMP()* that records a sensory-motor sample of the form given by Equation (45), where the visual position $\mathbf{v}_f$ is implemented as global variable and the DOF positions $\mathbf{p}$ are provided through *motor_in*. Both visual sample and motor sample are updated through ROS callback functions. Algorithm 6 includes three sub-functions that move the DOF:

- *moveDOFup()*, see Algorithm 7
- *moveDOFtoHP()*, see Algorithm 8
- *moveDOFdown()*, see Algorithm 9

The two sub-functions *moveDOFup()* and *moveDOFdown()* each contain a P control method in order to generate a new motor sample each time the functions are called. They also contain the auxiliary function *getDeltaDOF()* described by Algorithm 10. Its role is to compute the deviation l (expressed by the variable delta_dof in Algorithms 7 and 9) from the DOF home position h with $l > 0.0$. The computation is based on the given percentage Δ_{DOF} that denotes the percentage of position deviation from h. In particular, Δ_{DOF} describes the fraction of motion of the DOF that has the greatest range of all DOFs of the limb. The deviation l is needed for the sub-functions *moveDOFup()* and *moveDOFdown()* in order to move each DOF of the limb to the position $h + l$ and $h - l$, respectively. The parameter Δ_{DOF} was set to 5, i.e. the position deviation from the DOF home position was 5 % of the greatest total range that a DOF has on a particular limb. The parameter Δ_{DOF} was kept constant for both the exploration of the head and the arm joints. Algorithm 10 makes sure that the percentage of deviation is mapped to constant physical deviations $\pm l$ for all DOF of the limb being explored. This is achieved by different normalized values for the deviation $\pm l$ returned by the algorithm. In case of the NAO robot's head/neck joint with 2 DOF for instance, the yaw DOF has a greater range than the pitch DOF. Given $\Delta_{DOF} = 5$, Algorithm 10 computes a deviation of $l = 0.05$ for the yaw DOF and the $l = 0.1757$ for the pitch DOF. Due to the different physical ranges, these values result into a physical deviation of $\pm l = 11.95°$ from the home position, for both DOF. Hence, the explored range is $23.9°$ for the yaw and pitch DOF. This is relatively small (i.e. constrained) compared to the total (physical) range of the head/neck joint that is $239°$ for the yaw DOF and $68°$ for the pitch DOF according to the NAO robot's technical specification [174].
In case of the NAO robot's arm with 5 DOF[8], the same specified deviation $\Delta_{DOF} = 5$ is used for the exploration of the arm joints, where the greatest available range is $239°$ (shoulder pitch and elbow yaw [174]). The physical deviation from the home position is also $\pm l = 11.95°$ for each DOF of the arm.

[8] The fingers that are together regarded as 1 DOF are not taken into account.

Algorithm 7 Function *moveDOFup()* as part of Algorithm 6. Comments are in curly brackets.

Full name:
moveDOFup(int dof_idx*, double** motor_in*, double** motor_out*, double** hp*, int* num_dof*, int* limb*)*
Input: *int* dof_idx {index of the current DOF actuated},
int num_dof, limb {same as in Algorithm 6},
*double** motor_in, motor_out, hp {same as in Algorithm 6}
Output: *bool* result {indicating whether the DOF position $h + l$ has been reached}
Parameters: D {normalized difference between two consecutive angles sent},
K_p {gain for P control},
s_d {minimum difference between two consecutive angles sampled}
Variables: *bool* feature_lost, *static int* sample_idx, *static double* prev_motor_in,
static int prev_dof_idx,
double delta_dof {deviation $\pm l$ from home position h}

```
    {Initialize variables (lines 1 to 9)}
 1: feature_lost ← $false$
 2: result ← $false$
 3: sample_idx ← $0$
 4: prev_motor_in ← $0.0$
 5: prev_dof_idx ← $-1$
 6: delta_dof ← $0.0$
 7: if dof_idx $\neq$ prev_dof_idx then
 8:   prev_motor_in ← $0.0$
 9: end if
    {Check the visibility of visual feature $\mathbf{v}_f$ as global variable}
10: if $\mathbf{v}_f == (-1, -1, -1)^T$ then
11:   feature_lost ← $true$
12: end if
13: delta_dof ← getDeltaDOF(limb, dof_idx)   {Get delta_dof, given limb and dof_idx}
    {Generate motor command and update the output (lines 14 to 32)}
14: if motor_in[dof_idx] $<$ (hp[dof_idx] $+$ delta_dof) then
15:   $e_p$ ← | motor_out[dof_idx] $-$ motor_in[dof_idx] |   {Compute error $e_p$}
16:   motor_out ← motor_in $+ D + K_p \cdot e_p$   {Generate motor command by using P control}
    {Check the upper limit (lines 17 to 19)}
17:   if motor_out[dof_idx] $> 1.0$ then
18:     motor_out[dof_idx] ← $1.0$
19:   end if
20:   result ← $false$   {Indicate that position $h + l$ has not been reached yet}
    {Record samples if visual feature is visible (lines 21 to 25)}
21:   if (feature_lost $== false$) and (| motor_in[dof_idx] $-$ prev_motor_in | $> s_d$) then
22:     recordSampleSMP(sample_idx, motor_in, num_dof, limb)   {Record the current sample}
23:     sample_idx ← sample_idx $+ 1$
24:     prev_motor_in ← motor_in[dof_idx]
25:   end if
26: else
27:   result ← $true$   {Indicate that position $h + l$ has been reached}
28:   if feature_lost $== false$ then
29:     recordSampleSMP($-1$, motor_in, num_dof, limb)   {Record the end mark of sequence}
30:     sample_idx ← $0$
31:   end if
32: end if
33: prev_dof_idx ← dof_idx
34: return result
```

Algorithm 8 Function *moveDOFtoHP()* as part of Algorithm 6. Comments are in curly brackets.

Full name: *moveDOFtoHP(int* dof_idx*, double** motor_in*, double** motor_out*, double** hp*)*
Input: *int* dof_idx {index of the current DOF actuated},
*double** motor_in, motor_out, hp {same as in Algorithm 6}
Output: *bool* result {indicating whether the DOF home position h has been reached}
Parameters: ε_{DOF} {allowed deviation from a target DOF position}

```
1: result ← $false$
   {Move to home position $h$ if not there yet (lines 2 to 4)}
2: if (motor_in < (hp[dof_idx] − $\varepsilon_{DOF}$)) or (motor_in[dof_idx] > (hp[dof_idx] + $\varepsilon_{DOF}$)) then
3:    motor_out[dof_idx] ← hp[dof_idx]
4:    result ← $false$   {Indicate that home position $h$ has not been reached yet}
5: else
6:    result ← $true$   {Indicate that home position $h$ has been reached}
7: end if
8: return result
```

Algorithm 9 Function *moveDOFdown()* as part of Algorithm 6. Comments are in curly brackets.

Full name:
moveDOFdown(int dof_idx*, double** motor_in*, double** motor_out*, double** hp*, int* num_dof*, int* limb*)*
Input: {same as in Algorithm 7}
Output: *bool* result {indicating whether the DOF position $h - l$ has been reached}
Parameters: {same as in Algorithm 7}
Variables: {same as in Algorithm 7}

```
    {Initialize variables (lines 1 to 9)}
 1: feature_lost ← $false$
 2: result ← $false$
 3: sample_idx ← $0$
 4: prev_motor_in ← $0.0$
 5: prev_dof_idx ← $-1$
 6: delta_dof ← $0.0$
 7: if dof_idx $\neq$ prev_dof_idx then
 8:   prev_motor_in ← $0.0$
 9: end if
    {Check the visibility of visual feature $\mathbf{v}_f$ as global variable (lines 10 to 12)}
10: if $\mathbf{v}_f == (-1, -1, -1)^T$ then
11:   feature_lost ← $true$
12: end if
13: delta_dof ← getDeltaDOF(limb, dof_idx)   {Get delta_dof, given limb and dof_idx}
    {Generate motor command and update the output (lines 14 and 32)}
14: if motor_in[dof_idx] $>$ (hp[dof_idx] $-$ delta_dof) then
15:   $e_p$ ← | motor_out[dof_idx] $-$ motor_in[dof_idx] |   {Compute error $e_p$}
16:   motor_out ← motor_in $- D - K_p \cdot e_p$   {Generate motor command by using P control}
    {Check the upper limit (lines 17 to 19)}
17:   if motor_out[dof_idx] $< 0.0$ then
18:     motor_out[dof_idx] ← $0.0$
19:   end if
20:   result ← $false$   {Indicate that position $h + l$ has not been reached yet}
    {Record samples if visual feature is visible (lines 21 to 25)}
21:   if (feature_lost $== false$) and (| motor_in[dof_idx] $-$ prev_motor_in | $> s_d$) then
22:     recordSampleSMP(sample_idx, motor_in, num_dof, limb)   {Record the current sample}
23:     sample_idx ← sample_idx $+ 1$
24:     prev_motor_in ← motor_in[dof_idx]
25:   end if
26: else
27:   result ← $true$   {Indicate that position $h + l$ has been reached}
28:   if feature_lost $== false$ then
29:     recordSampleSMP($-1$, motor_in, num_dof, limb)   {Record the end mark of sequence}
30:     sample_idx ← $0$
31:   end if
32: end if
33: prev_dof_idx ← dof_idx
34: return result
```

Algorithm 10 Function *getDeltaDOF()* as part of Algorithms 7 and 9. Comments are in curly brackets.

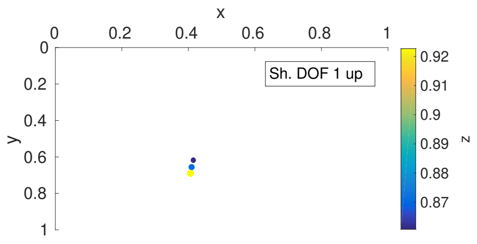
Full name: *getDeltaDOF(int* limb*, int* dof_idx*)*
Input:
int limb {number representing the body part for exploration, i.e. either head, left arm, or right arm; limb $== 0$: head/neck joints; limb $== 1$: left arm joints; limb $== 2$: right arm joints},
int dof_idx {index of the current DOF actuated}
Output: *double* result {the normalized value of delta_dof}
Parameters: Δ_{DOF} {percentage of position deviation from h}
Variables:
double range_dof[], delta_dof[], dof_limit_up[], dof_limit_lo[] {array size in each case: the number of DOF of the limb},
double max_range, phys_delta_dof
Prerequisites: physical upper limit and physical lower limit (in radians) of each DOF to be explored

```
     {Store the physical DOF limits of the limb (lines 1 to 4)}
 1:  for each DOF $i$ of the limb do
 2:      dof_limit_up[$i$] $\leftarrow$ physical upper limit of DOF $i$ of the limb
 3:      dof_limit_lo[$i$] $\leftarrow$ physical lower limit of DOF $i$ of the limb
 4:  end for
     {Compute the total physical range (in radians) per DOF of the limb (lines 5 to 7)}
 5:  for each DOF $i$ of the limb do
 6:      range_dof[$i$] $\leftarrow$ | dof_limit_up[$i$] $-$ dof_limit_lo[$i$] |
 7:  end for
 8:  $i_g \leftarrow$ index of the DOF with the greatest range among all DOFs of the limb
 9:  max_range $\leftarrow$ range_dof[$i_g$]
     {Compute the fraction of the physical range determined by $\Delta_{DOF}$ (line 10)}
10:  phys_delta_dof $\leftarrow \Delta_{DOF} \cdot$ max_range $/ 100.0$
     {Compute the normalized delta_dof of the current DOF actuated (lines 11 to 14)}
11:  for each DOF $i$ of the limb do
12:      delta_dof[$i$] $\leftarrow$ phys_delta_dof $/$ range_dof[$i$]
13:  end for
14:  result $\leftarrow$ delta_dof[dof_idx]
15:  return result
```

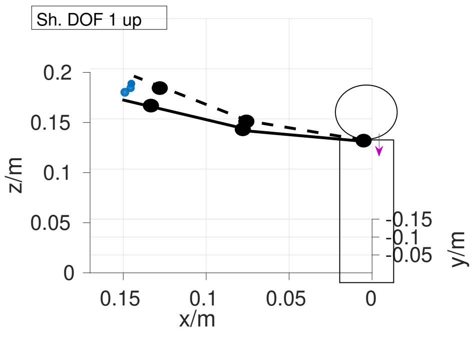
4.4.3. Experiment: Constrained DOF Exploration on the NAO Robot

The constrained DOF exploration was conducted for the right arm of the NAO robot, with the head at a fixed pose such that the visual marker attached on the wrist is in the robot's FOV. The exploration was conducted with the following parameters: $\Delta_{DOF} = 5$, $D = 0.02$, $K_p = 3$, $s_D = 0.001$, and $\varepsilon_{DOF} = 0.03$.

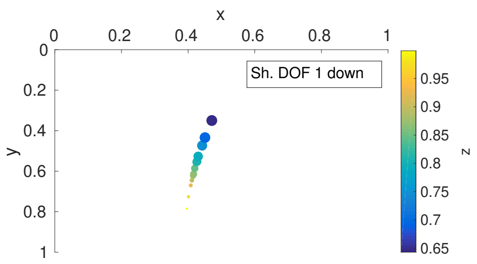
This exploration process is visualized in the Figures 27, 28, 29. In these figures, the FOV shows the recorded trace of the arm tip marker, the corresponding motor patterns, i.e. DOF positions, are also recorded (not shown). The idle sequence (not shown) consists of 4 sensory-motor samples of the arm tip in home position. In total, 11 sequences (2 sequences per DOF and the additional idle sequence) were collected. The total number of samples was 131. It can be observed that the recorded sequences in "down" direction contain more samples than those in the "up" direction. This is due to the value of ε regulating the position tolerance when the DOF is commanded into its home position. It could be observed that with $\varepsilon = 0.03$, approximately one-third of the samples were recorded when the DOF was still in home position. Nevertheless, the sequence recorded in the "up" direction does *not* have to be of the same length than the sequence recorded in the "down" direction. Sequences recorded in "up" and "down" direction can be indeed different in their respective lengths. Important is the motion of the DOF and the resultant changes in the perception of the visual feature.



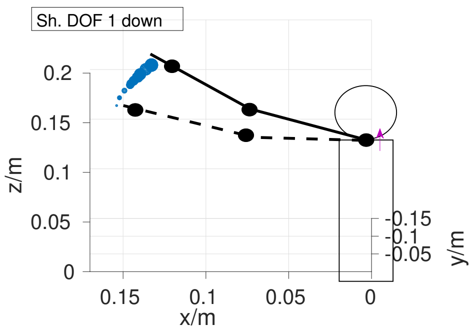
(a) FOV, Sh. DOF 1 up, 9 collected samples



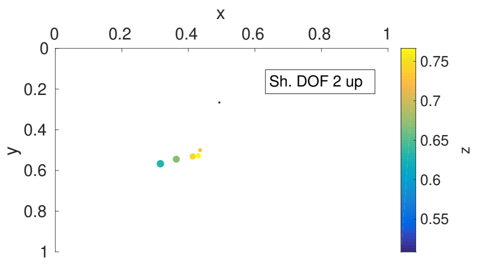
(b) Scenario 27(a) in torso frame



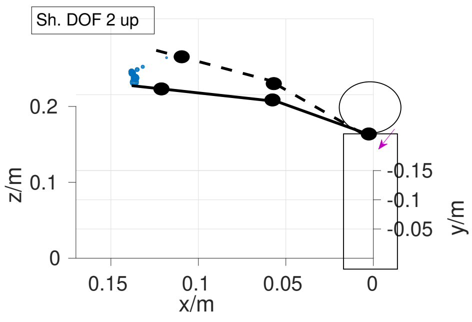
(c) FOV, Sh. DOF 1 down, 21 collected samples



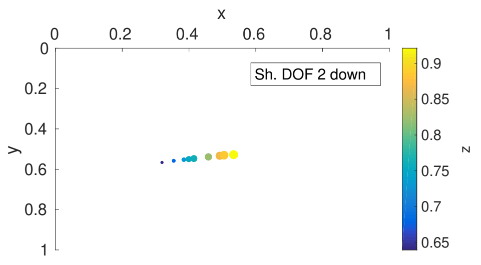
(d) Scenario 27(c) in torso frame



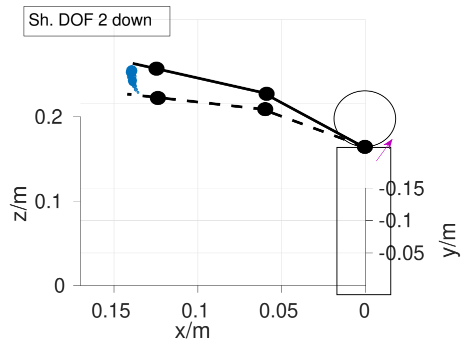
(e) FOV, Sh. DOF 2 up, 10 collected samples



(f) Scenario 27(e) in torso frame

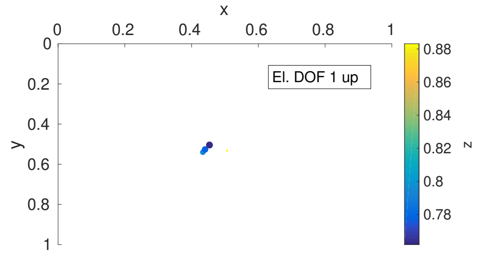


(g) FOV, Sh. DOF 2 down, 15 collected samples

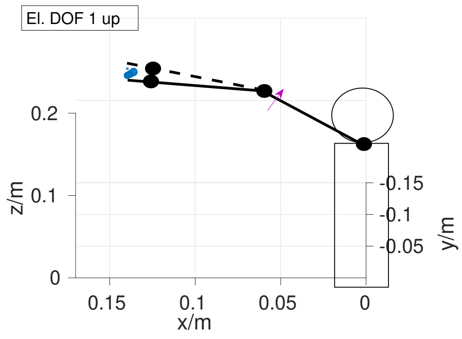


(h) Scenario 27(g) in torso frame

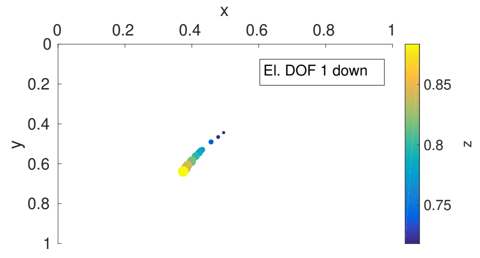
Figure 27 Constrained DOF exploration applied to the 5 DOF right arm of the NAO robot, 2 DOF *shoulder* joint (abbreviated by "Sh."). Visual samples (arm tip position) in the robot's FOV on the left, with the distance of each sample shown by the colour bar. The same samples in the torso frame on the right. Temporal information is conveyed by the circle size, with the smallest circle at motion onset and the biggest at the end of motion. In the torso frame, the initial arm pose (before motion onset) is shown by the dashed line and the end pose by the solid line. The cases "up" and "down" represent the angle position of each DOF actuated, not the geometrical direction.



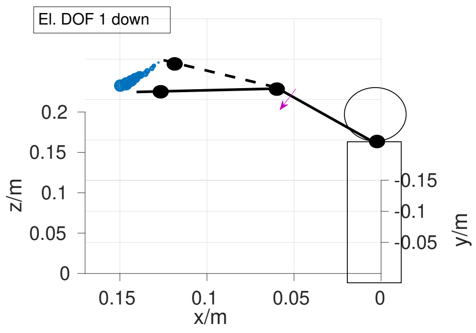
(a) FOV, El. DOF 1 up, 9 collected samples



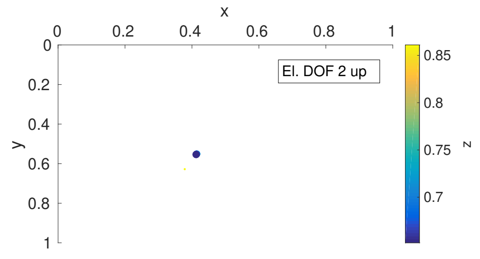
(b) Scenario 28(a) in torso frame



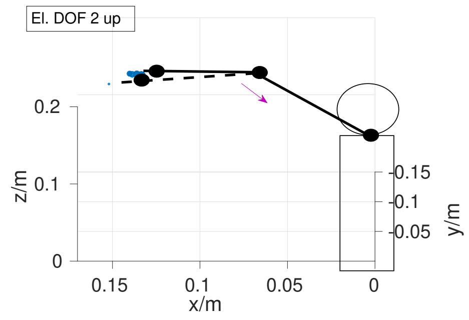
(c) FOV, El. DOF 1 down, 21 collected samples



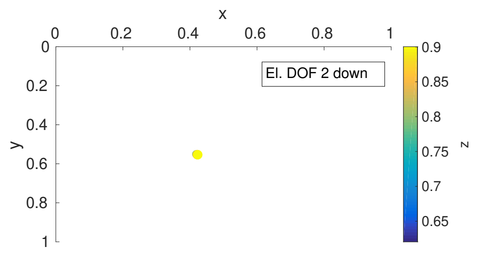
(d) Scenario 28(c) in torso frame



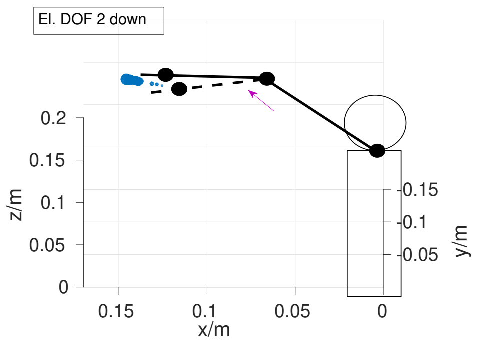
(e) FOV, El. DOF 2 up, 11 collected samples



(f) Scenario 28(e) in torso frame

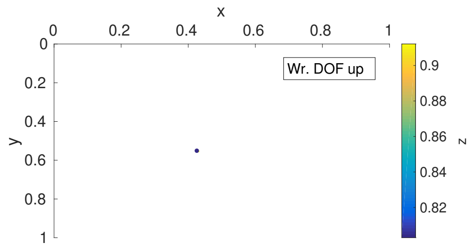


(g) FOV, El. DOF 2 down, 18 collected samples

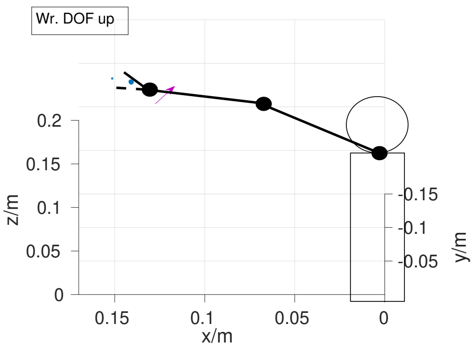


(h) Scenario 28(g) in torso frame

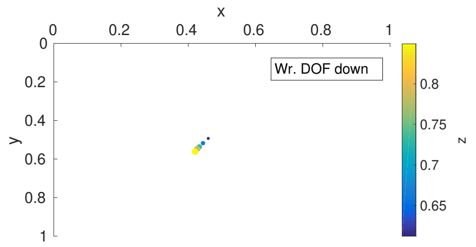
Figure 28 Continued: Constrained DOF exploration applied to the 5 DOF right arm of the NAO robot, 2 DOF *elbow* joint (abbreviated by "El."). See caption of Figure 27 for information about the visual samples.



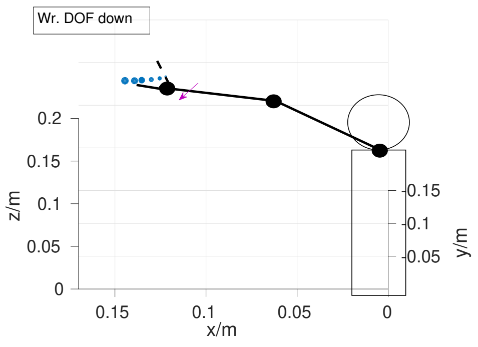
(a) FOV, Wr. DOF 1 up, 4 collected samples



(b) Scenario 29(a) in torso frame



(c) FOV, Wr. DOF 1 down, 9 collected samples

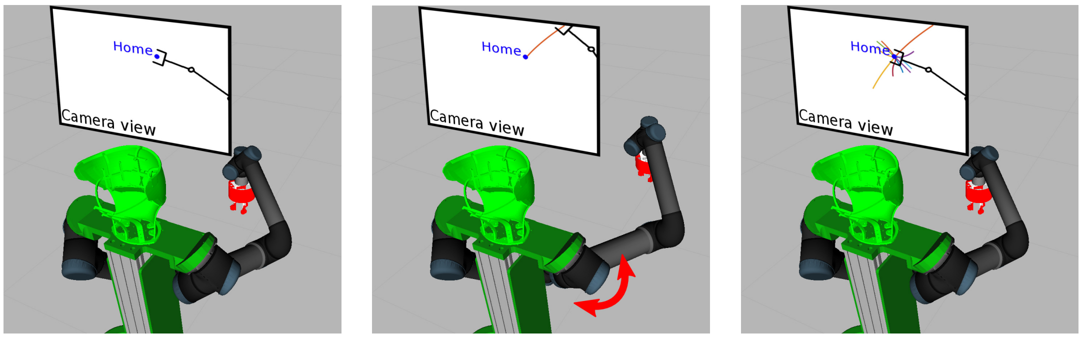


(d) Scenario 29(c) in torso frame

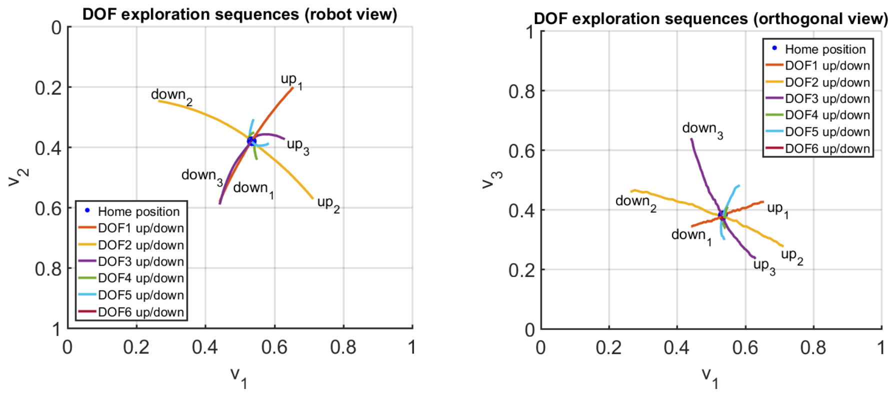
Figure 29 Continued: Constrained DOF exploration applied to the 5 DOF right arm of the NAO robot, 1 DOF *wrist* joint (abbreviated by “Wr.”). See caption of Figure 27 for information about the visual samples.

4.4.4. Experiment: Constrained DOF Exploration on the TOMM Robot

Constrained DOF exploration was also conducted for the 6 DOF right arm of the TOMM robot [52]. Figure 30 illustrates snapshots of the exploration procedure on the robot, along with the robot's FOV. On TOMM, each DOF was moved for $\Delta_{DOF} = 10$, i.e. 10 % of the total



(a) Home position (b) Moving shoulder DOF (c) All DOF explored



(d) Recorded sequences of the arm tip (robot view) (e) Recorded sequences of the arm tip (orthogonal view)

Figure 30 Constrained DOF exploration on the TOMM robot. The experimental outcome was visualized by Burger in [34]. The right arm with 6 DOF was explored. The robot starts from home position 30(a) and explores each DOF one by one 30(b) until every DOF of the limb has been explored 30(c). In 30(a), 30(b), 30(c), the robot's FOV (white plane) shows the trace of the arm tip corresponding to each situation of the exploration process. All these traces are shown in 30(d), 30(e), where v_1, v_2, v_3 are the normalized image coordinates with v_3 denoting the distance. The traces of the arm tip and the corresponding DOF positions form the training data for the PAS.

available motion range, yielding an exploration of 20 % of the full DOF range. Given 6 DOF, 13 sequences (2 times 6 plus the idle sequence) have been recorded, each sequence with up to 29 samples. The total number of recorded samples was 349.

4.5. Generalization from Motion Patches to Meaningful Behaviour

The previous Sections 4.3 and 4.4 have provided answers to the first two questions deduced from the poverty of stimulus (Section 4.2.1). This section continues to answer the third question: How can a new action selection method make sense of the limited set of samples collected during the constrained DOF exploration? In other words, how can the method generalize from a minimum amount of samples? Note that this is priority number 1 of the goals realized by PAS (Table 16).
This leads to the following problem: generalization from motion patches (constrained DOF exploration) to a coordinated skill. A coordinated skill is a sensory-motor skill that is *not* task-centric. Although not task-centric, it has to appear as a sort of meaningful behaviour that can be part of a hierarchy of behaviours. For example, head motions to bring an object into the centre of the FOV, arm motion to reach an object, head motion to bring an object out of the FOV that can be interpreted as disengaging in attention (i.e. refocusing on something new), etc. Strongly related to such coordination skills is the aspect of robustness, see priority number 3 of the goals (Table 16). Visual features can become partly occluded and are temporarily not available for processing. Thus, the new method has to integrate both generalization ability and robustness.

4.6. Components of the PAS

In order to solve the problem of generalization (Section 4.5), I propose an action selection method that is inspired by the prediction capability of the cortex.
Several cortical regions are involved in sensory-motor coordination: the medial superior temporal (MST) area for the prediction of self-motion and object motion [234], [224], as well as the dorsal premotor (PMd) area and primary motor (M1) area for response selection [16].
Moreover, a computational model for the prediction of self-motion and object motion is aimed at the realization of robustness to temporal loss of sensory data. Particularly, the observation and subsequent prediction of object motion aims at the realization of goal priority 2 (Table 16).
Computationally, the prediction of self-motion and the prediction of object motion are each modelled through neural network-based algorithms that use the EO-MTRNN proposed in Chapter 3. Figure 31 gives an overview on the components of the PAS and how they are linked together. As shown by the figure, the main components are the self-motion predictor (SMP) and the feature predictor (FP). The output of the SMP and FP is fed into the action selection algorithm referred to as "response selection" and "motor command calculation". The motor command calculation yields the proprioceptive pattern $\mathbf{p}_{out}$ sent to the robot's limb that is to be controlled.

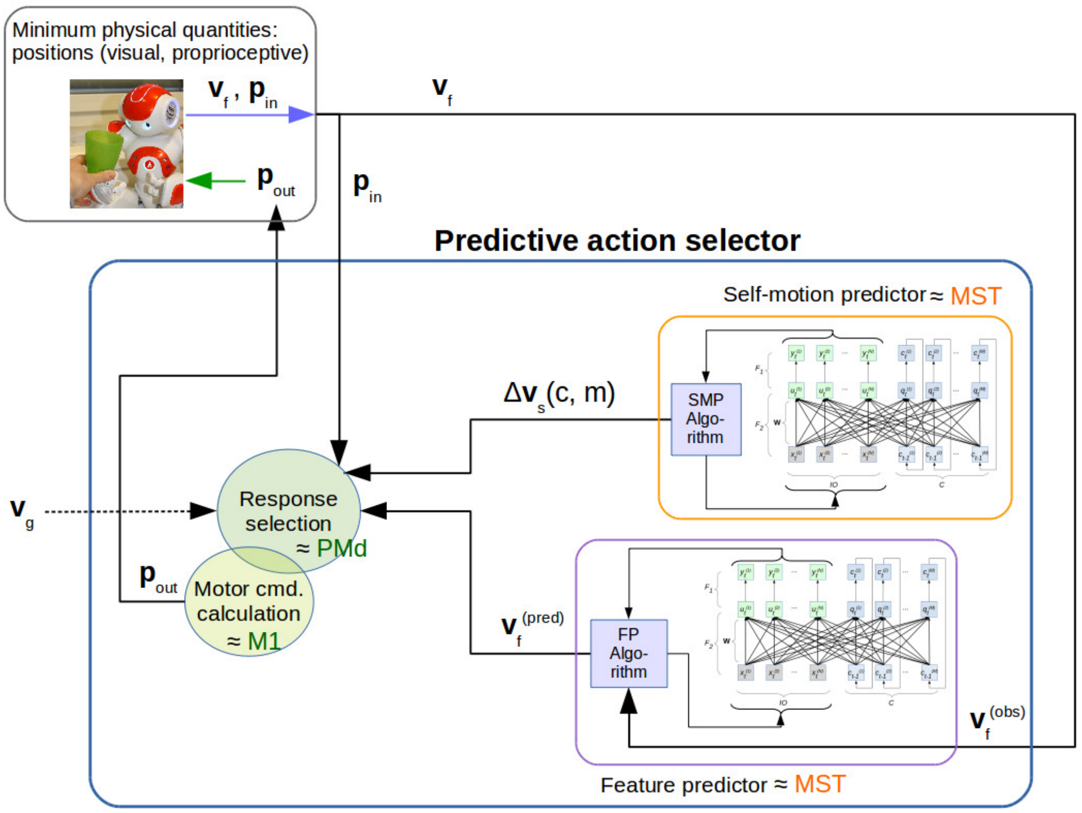

Minimum physical quantities:
positions (visual, proprioceptive)
$\mathbf{v}_f$, $\mathbf{p}_{in}$
$\mathbf{p}_{out}$
$\mathbf{v}_f$
$\mathbf{p}_{in}$
Predictive action selector
Self-motion predictor ≈ MST
SMP Algo-rithm
$\Delta\mathbf{v}_s$(c, m)
Response selection ≈ PMd
$\mathbf{v}_g$
$\mathbf{p}_{out}$
Motor cmd. calculation ≈ M1
$\mathbf{v}_f^{(pred)}$
FP Algo-rithm
$\mathbf{v}_f^{(obs)}$
Feature predictor ≈ MST

Figure 31 Components of the PAS. The PAS (blue-framed transparent box) contains two neural network-based components: the self-motion predictor (SMP) and the feature predictor (FP). Both use the EO-MTRNN proposed in Chapter 3 and emulate the involvement of the MST in the prediction of self-motion and object motion. The SMP computes the visual consequence of the predicted self-motion $\Delta\mathbf{v}_s$ and the FP computes the predicted position $\mathbf{v}_f$ of an external object. The signals from SMP and FP are sent to the action selection algorithm "response selection" emulating the involvement of the PMd in response selection. Part of that algorithm is the "motor command calculation" emulating the involvement of M1 in signal generation for voluntary movements. The resulting proprioceptive pattern $\mathbf{p}_{out}$ is sent to the robot actuating the limb that is controlled.

4.6.1. Input and Output of the PAS

The input and output of the PAS follows the Figures 23 and 31.
In particular, the input is:

- Visual goal $\mathbf{v}_g$: vector containing the normalized 3D position of a goal point in the robot's FOV
- Visual feature $\mathbf{v}_f$: vector containing the normalized 3D position of a visual feature of interest, i.e. the arm tip marker or an external object
- Proprioceptive input $\mathbf{p}_{in}$: vector containing the normalized DOF positions of the limb that is to be controlled, i.e. head or arm

The output is proprioceptive pattern $\mathbf{p}_{out}$ that is a vector containing the normalized target DOF positions of the limb. The vector $\mathbf{p}_{out}$ is de-normalized to physical DOF positions and sent to the robot.

4.6.2. Self-Motion Predictor

In each computational cycle, i.e. for each proprioceptive pattern $\mathbf{p}_{out}$ that is going to be computed, the SMP predicts the change of the visual feature of interest $\Delta\mathbf{v}_s$ as a result of self-motion. The change $\Delta\mathbf{v}_s$ is caused by self-motion only.
As shown in Figure 31, the SMP uses the proposed EO-MTRNN (Chapter 3). Prior to its prediction ability, the EO-MTRNN of the SMP has to be trained with the visuo-motor sequences generated during the constrained DOF exploration (Section 4.4). It will be trained with all samples generated during the exploration of the body part that the PAS learns to coordinate, i.e. either head or arm. Every learned sequence is represented by the index c that corresponds to the initial potential state of the context neurons of the EO-MTRNN.
In Figure 31, the box "Self-motion predictor" contains the SMP algorithm and the EO-MTRNN. The purpose of the SMP algorithm (see Algorithm 11) is to run the EO-MTRNN in a closed-loop mode, in which every predicted sample is used as input sample at the next time step. The prediction starts with the initial sample $\mathbf{s}_0^T = (\mathbf{v}_0^T \quad \mathbf{p}_0^T)$ of the sequence c recalled by the EO-MTRNN. The algorithm controls the prediction length by the parameter m that determines how many samples will be predicted ahead. Note that the EO-MTRNN predicts visuo-motor samples $\mathbf{s}_i$ of the form $\mathbf{s}_i^T = (\mathbf{v}_i^T \quad \mathbf{p}_i^T)$ with i as discrete time step. Here, $i \in [0, m]$. Thus, for every predicted visual state $\mathbf{v}_i$, there exists a corresponding motor state $\mathbf{p}_i$ that is needed for the subsequent computation of the target motor state by the action selection method (Section 4.6.4).
The SMP contains an additional prediction mode that is termed second-stage mode (see Algorithm 12). It is used for the second-stage prediction mode of the PAS that will be explained later in this chapter.

Algorithm 11 Self-motion predictor using the function *predictNextSample()* of the EO-MTRNN $M1$ to recall sequences from a previous constrained DOF exploration. This algorithm realizes the box "Self-motion predictor" of Figure 31. The algorithm output is later used to compute $\Delta\mathbf{v}_s$ according to Equation (46) and to compute $\Delta\mathbf{p}$ according to Equation (52).

Requirements:
EO-MTRNN $M1$ trained with the sequences obtained by the constrained DOF exploration
Input: c {Index representing the initial context potentials of the sequence to be recalled},
m {Length (integer) of the predicted sequence}
Output: latest sample $\mathbf{s} := \mathbf{s}_{m-1}^T = (\mathbf{v}_{m-1}^T \quad \mathbf{p}_{m-1}^T)$ of the predicted sensory-motor sequence of length m

```
1: $\mathbf{s} \leftarrow M1$.getInitialSample($c$)
2: for $i \leftarrow 0; i < m; i \leftarrow i + 1$ do
3:     $\mathbf{s} \leftarrow M1$.predictNextSample($\mathbf{s}$, $c$)
4: end for
5: return $\mathbf{s}$
```

Algorithm 12 Self-motion predictor (second-stage mode) using the function *predictNextSample()* of the EO-MTRNN $M1$ to predict samples based on learned sensory-motor sequences from a previous interaction phase, e.g. reaching. This algorithm is only used for the *second-stage prediction mode* of the PAS (Section 4.7.2).

Requirements:
EO-MTRNN $M1$ trained with the sequences obtained by an interaction phase
Input: sample $\mathbf{s}_t^T = (\mathbf{v}_t^T \quad \mathbf{p}_t^T)$ at any time step t,
c {Index representing the initial context potentials of the sequence to be recalled}
Output: a sample $\mathbf{s}_{t+1}$ of a one-step prediction $t + 1$

```
1: $\mathbf{s}_{t+1} \leftarrow M1$.predictNextSample($\mathbf{s}_t$, $c$)
2: return $\mathbf{s}_{t+1}$
```

4.6.3. Feature Predictor

As the second neural network-based algorithm of the PAS, the feature predictor learns the dynamics of *external* entities like objects by observing them. Note that this is goal priority number 2 of Table 16. In other words, the feature predictor can learn the trajectory of external objects the robot is supposed to interact with. An example scenario is a moving cup that the robot is tracking by head/eye motions and is reaching for. When the cup gets occluded by another object, the feature predictor computes the locations where the object may reappear. Hence, this mechanism increases the robustness of the interaction by compensating for the temporal loss of visual features.

The visual feature extractor (Section 4.3.2) outputs the current 3D position $\mathbf{v}_f$ of the visual feature. If the *observed* $\mathbf{v}_f := \mathbf{v}_f^{(obs)}$ gets lost, the feature predictor computes a *predicted* $\mathbf{v}_f := \mathbf{v}_f^{(pred)}$ based on the last observed samples of positions. Algorithm 13 describes the working principle that contains a sequence recognition function (Algorithm 14) using the root-mean-square deviation (RMSD) metric.

4.6.4. Action Selection Method Integrating the Neural Components

The SMP and FP outputs are parts of the action selection method that models the PMd area overlapping with the M1 area. Neuroscientific evidence highlights the crucial role of the PMd area. The PMd area processes information for the visual guidance of arm motion [46], [145]. In addition, it selects motor programs based on learned associations [46], [225], [47], [16]. This neuroscientific evidence, in particular the overlap of PMd area with M1 area, suggests to integrate action selection selection and action generation into one computational framework that is listed as goal priority number 4 (Table 16).

The proposed action selection algorithm uses the visual part of the latest predicted sample delivered by Algorithm 11 in order to determine the predicted change $\Delta\mathbf{v}_s$. The change $\Delta\mathbf{v}_s$ depends on the sequence c and the prediction length m. Mathematically, this is expressed by

$$\Delta\mathbf{v}_s(c,m) = \mathbf{v}_s(c,m) - \mathbf{v}_s(c,0) \tag{46}$$

where the vector $\mathbf{v}_s(c,m)$ is the *predicted* visual sample of the learned *sequence* c at time step m, given its initial sample $\mathbf{v}_s(c,0)$. Note that the index s refers to the influence by self-motion only. Equation (46) describes two different scenarios depending on the body part that is controlled:

1. If the PAS learns to coordinate the head, the constrained DOF exploration has been conducted for the head, with $\mathbf{v}_f$ representing the target object position. In that case, $\Delta\mathbf{v}_s$ describes how the position of a target object would shift in the robot's FOV as a result of a particular head motion.

2. If the PAS learns to coordinate the arm, the constrained DOF exploration has been conducted for the arm, with $\mathbf{v}_f$ representing the arm tip position. In that case, $\Delta\mathbf{v}_s$ describes how the position of the arm tip would shift in the robot's FOV as a result of a particular arm motion.

Algorithm 13 Feature predictor. Comments are in curly brackets.

```
Requirements: EO-MTRNN $M2$ trained with $N$ different sequences of $\mathbf{v}_f$
Input: observed $\mathbf{v}_f^{(obs)}$ delivered by the visual feature extractor (Algorithm 4.3.2)
Output: predicted $\mathbf{v}_f^{(pred)}$
Parameters: $R$   {observation length (integer), i.e. length of the observed sequence used for
                    recognition},
            $P$   {prediction length (integer), i.e. length of the predicted sequence}
Variables: $\mathbf{B}^{(obs)}$   {2D array, size $3 \times R$, containing samples of $\mathbf{v}_f^{(obs)}$},
           $\mathbf{B}^{(pred)}$   {2D array, size $3 \times R$, containing samples of $\mathbf{v}_f^{(pred)}$},
           $\mathrm{s}$   {3D vector to temporarily store samples of $\mathbf{v}_f$},
           $\mathrm{s}^*$   {3D vector to temporarily store $\mathbf{v}_f^{(pred)}$ predicted by $M2$},
           $x$   {sample index, e.g. $\mathbf{B}^{(pred)}\big|_{x=0}$ means the first sample of $\mathbf{B}^{(pred)}$},
           $q_w$   {index of a learned sequence that most closely matches the one observed}

 1: if $\mathbf{B}^{(obs)}$ contains entries of $\mathbf{0}$ and $\mathbf{v}_f$ is observable then
 2:   Fill $\mathbf{B}^{(obs)}$ with the first $R$ samples of the observed $\mathbf{v}_f$
 3: else
 4:   if $\mathbf{v}_f$ is observable then
      {Update $\mathbf{B}^{(obs)}$ with latest sample $\mathbf{v}^{(obs)}$ and shift the remaining samples, "drop" the oldest (lines 5
      to 8)}
 5:     for $i \leftarrow 1; i < R; i \leftarrow i + 1$ do
 6:       $\mathbf{B}^{(obs)}\big|_{x=i-1} \leftarrow \mathbf{B}^{(obs)}\big|_{x=i}$
 7:     end for
 8:     $\mathbf{B}^{(obs)}\big|_{x=R-1} \leftarrow \mathbf{v}_f^{(obs)}$
 9:   else
      {Update $\mathbf{B}^{(obs)}$ with latest sample $\mathbf{v}^{(pred)}$ stored in $\mathrm{s}$ and shift the remaining samples, "drop" the
      oldest (lines 10 to 13)}
10:     for $i \leftarrow 1; i < R; i \leftarrow i + 1$ do
11:       $\mathbf{B}^{(obs)}\big|_{x=i-1} \leftarrow \mathbf{B}^{(obs)}\big|_{x=i}$
12:     end for
13:     $\mathbf{B}^{(obs)}\big|_{x=R-1} \leftarrow \mathrm{s}^*$
14:   end if
15:   $q_w \leftarrow$ recognizeSequence($\mathbf{B}^{(obs)}$, $\mathbf{B}^{(pred)}$)   {Recognize the sequence, Algorithm 14}
      {Predict the recognized sequence (lines 16 to 19)}
16:   $\mathrm{s} \leftarrow \mathbf{B}^{(obs)}\big|_{R-1}$
17:   for $i \leftarrow 0; i < P; i \leftarrow i + 1$ do
18:     $\mathrm{s} \leftarrow M2$.predictNextSample($\mathrm{s}$, $q_w$)
19:   end for
20: end if
21: $\mathrm{s}^* \leftarrow \mathrm{s}$
22: return $\mathrm{s}$
```

Algorithm 14 Function *recognizeSequence(...)* as part of Algorithm 13. Comments are in curly brackets.

```
Requirements: {same as in Algorithm 13}
Input: $\mathbf{B}^{(obs)}$    {2D array, size $3 \times R$, containing samples of $\mathbf{v}_f^{(obs)}$},
       $\mathbf{B}^{(pred)}$    {2D array, size $3 \times R$, containing samples of $\mathbf{v}_f^{(pred)}$}
Output: $q_w$    {index of a learned sequence that most closely matches the one observed}
Parameters: $R$    {observation length (integer), same as in Algorithm 13}
Variables: $\mathbf{d}$    {array (of floating-points) of size $N$},
           $\mathbf{s}$    {3D vector to temporarily store samples of $\mathbf{v}_f$}

1: for each sequence $q$ learned by $M2$ do
2:   for $i \leftarrow 0; i < R; i \leftarrow i + 1$ do
3:     if $i == 0$ then
4:       $\mathbf{B}^{(pred)}\big|_{x=0} \leftarrow \mathbf{B}^{(obs)}\big|_{x=0}$    {Initialize first sample of the predicted sequence}
5:     end if
6:     $\mathbf{s} \leftarrow \mathbf{B}^{(obs)}\big|_{x=i}$    {Read sample from buffer}
7:     $\mathbf{s} \leftarrow M2$.predictNextSample($\mathbf{s}$, $q$)    {Predict next sample}
8:     if $i < (R - 1)$ then
9:       $\mathbf{B}^{(pred)}\big|_{x=i+1} \leftarrow \mathbf{s}$    {Store predicted sample into buffer}
10:    end if
11:  end for
   {Calculate the discrepancy between $\mathbf{B}^{(obs)}$ and $\mathbf{B}^{(pred)}$ using the RMSD metric (line 12)}
12:  $\mathbf{d}[q] \leftarrow \sqrt{\frac{1}{R}\sum_{i=0}^{R-1}(\mathbf{B}^{(obs)}\big|_{x=i} - \mathbf{B}^{(pred)}\big|_{x=i})^2}$
13:  $\mathbf{B}^{(pred)} \leftarrow \mathbf{0}$    {Empty $\mathbf{B}^{(pred)}$}
14: end for
15: $q_w \leftarrow \min(\mathbf{d})$    {Select the sequence with the minimum discrepancy}
16: return $q_w$
```

In each of the two aforementioned scenarios, the particular motion is represented by (c, m). The motion is imaginary (i.e. predicted by the SMP), it is characterized by its initial state $\mathbf{v}_s(c, 0)$ and its end state $\mathbf{v}_s(c, m)$.

The biological analogy is that $\mathbf{v}_f$ and $\Delta\mathbf{v}_s(c, m)$ correspond to the contribution of MST regions of the visual pathway involved in visuo-motor coordination. Especially $\Delta\mathbf{v}_s$ describes the predicted perception of self-motion relative to a (visual) feature of interest [234].

The parameters (c, m) represent a motion sequence where only one DOF is active while the others stay in constant position. The action selection method combines the contribution of each DOF to determine the target motor positions $\mathbf{p}_{out}$.

A key ingredient of the proposed method is the following value function for action selection:

$$V(c, m) = ||\mathbf{v}_g + a_f \cdot \mathbf{v}_f + a_s \cdot \Delta\mathbf{v}_s(c, m)|| \tag{47}$$

The notation $||(...)||$ is the Euclidean norm. In general, given a vector $\mathbf{x} = (x_1, x_2, ..., x_n)^T$, then $||\mathbf{x}|| = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$. Here, the vector dimension is $n = 3$ due to the 3D visual space.

The scalars a_f and a_s are termed alteration parameters. They support an integration of multiple sensory input [41]. Here, they also bias the robot's behaviour. A switch of the sign of a_s toggles between minimizing and maximizing the predicted visual error the goal $\mathbf{v}_g$. For example, when the PAS coordinates the head or eye motions, a_s can toggle between an object tracking behaviour and a behaviour where the robot avoids looking at the object (i.e.

object evasion) [220]. The default setting is $a_f = -1$ and $a_s = -1$. Hence, Function (47) describes the predicted deviation in the visual space between the goal $\mathbf{v}_g$ and the feature of interest $\mathbf{v}_f$, depending on the robot's own motion. In case of head control, if $\mathbf{v}_f$ becomes temporarily not observable due to occlusion or covering of the object, the FP (Section 4.6.3) will provide its predicted position based on the latest samples observed. The same prediction can be used as goal position for the arm control, i.e. $\mathbf{v}_g^{(A)} := \mathbf{v}_f^{(H)}$ if a PAS for head control and a PAS for arm control are simultaneously active.
Based on Function (47), the action selection is formulated as

$$\min V(c, m) = \min ||\mathbf{v}_g + a_f \cdot \mathbf{v}_f + a_s \cdot \Delta\mathbf{v}_s(c, m)|| \tag{48}$$

given a set $C = \{c_1, c_2, ..., c_n\}$ of n learned sequences. Note that these are the sequences generated by the constrained DOF exploration before and have been learned by the SMP. Since the number of learned sequences is small and m is known and constant, Equation (48) is solved by computing V for each element of C. Then, the winner sequence c_w is selected, that corresponds to the smallest value of V computed.

4.6.4.1 Procedure of Predictive Action Selection

For each DOF that is controlled, the method executes the following Steps (49)–(55):
The method computes the visual error e_v.

$$e_v = ||\mathbf{v}_g - \mathbf{v}_f|| \tag{49}$$

Based on the visual error, the method determines the prediction length m by

$$m = \lfloor \kappa \cdot e_v \rfloor \tag{50}$$

where κ is a constant termed prediction length factor.
Given m, the method selects the action by

$$\min V(c, m). \tag{51}$$

In other words, it computes the self-motion effects $\Delta\mathbf{v}_s$ and determines the visuo-proprioceptive sequence c_w that minimizes the Function (47).
Given c_w and m, the method computes the new DOF position by

$$\Delta p(c_w, m) = p(c_w, m) - p(c_w, 0) \tag{52}$$

$$p_{out} = p_{in} + \Delta p(c_w, m) \tag{53}$$

where $p(c_w, m)$ is the predicted DOF position extracted from the visuo-proprioceptive winner sequence c_w at time step m.
After determining p_{out}, the method computes the resulting visual feature $\mathbf{v}_f^*$ that is caused by

the predicted change $\Delta \mathbf{v}_s(c_w, m)$, once p_{out} has been commanded to the DOF.

$$\mathbf{v}_f^* = \mathbf{v}_f + \Delta \mathbf{v}_s(c_w, m) \tag{54}$$

Finally, the method updates the input visual feature $\mathbf{v}_f$.

$$\mathbf{v}_f \leftarrow \mathbf{v}_f^* \tag{55}$$

The entire procedure is also summarized by Figure 32.

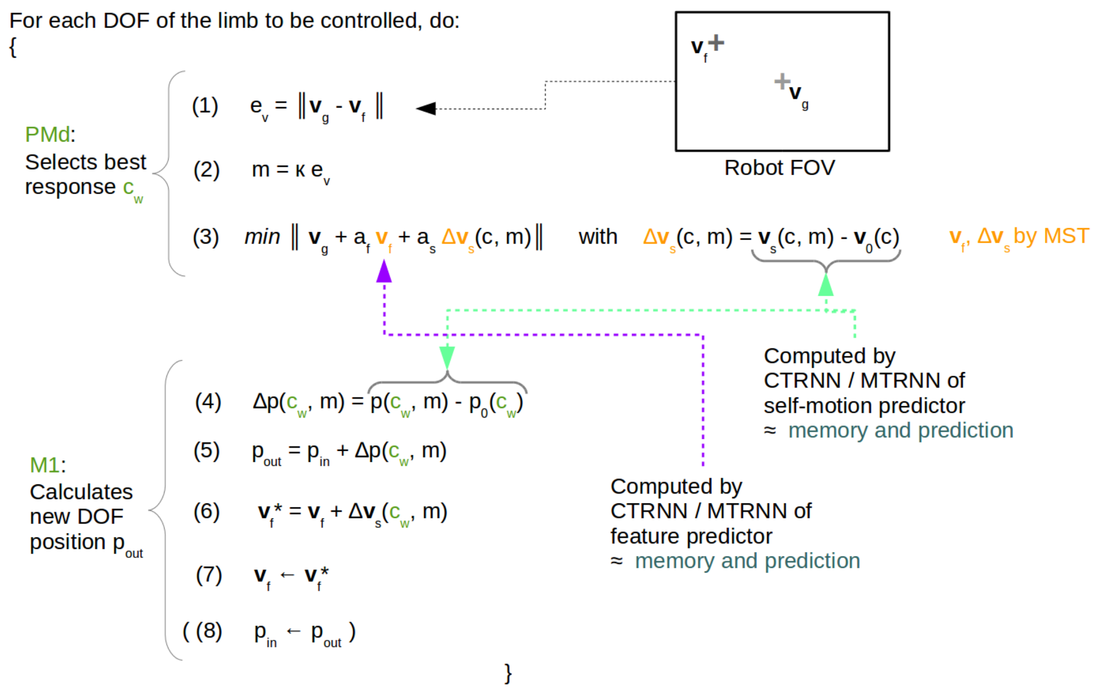

For each DOF of the limb to be controlled, do:
{
PMd: Selects best response c_w
(1) $e_v = \| \mathbf{v}_g - \mathbf{v}_f \|$
(2) $m = \kappa e_v$
(3) $min \| \mathbf{v}_g + a_f \mathbf{v}_f + a_s \Delta\mathbf{v}_s(c, m) \|$ with $\Delta\mathbf{v}_s(c, m) = \mathbf{v}_s(c, m) - \mathbf{v}_0(c)$
$\mathbf{v}_f$, $\Delta\mathbf{v}_s$ by MST
$\mathbf{v}_f$
$\mathbf{v}_g$
Robot FOV
Computed by CTRNN / MTRNN of self-motion predictor ≈ memory and prediction
M1: Calculates new DOF position p_{out}
(4) $\Delta p(c_w, m) = p(c_w, m) - p_0(c_w)$
(5) $p_{out} = p_{in} + \Delta p(c_w, m)$
(6) $\mathbf{v}_f^* = \mathbf{v}_f + \Delta\mathbf{v}_s(c_w, m)$
(7) $\mathbf{v}_f \leftarrow \mathbf{v}_f^*$
((8) $p_{in} \leftarrow p_{out}$)
Computed by CTRNN / MTRNN of feature predictor ≈ memory and prediction
}

Figure 32 Illustration of the predictive action selection. The orange and dark green parts refer to the biological inspiration, see also the abstraction in Figure 31. Note that step (8) is optional, i.e. the feedback of proprioception is only needed for mental simulation. The neural network of the SMP and FP does not necessarily have to be a MTRNN. A CTRNN (i.e. single timescale context) may also learn the sequences, since they are relatively short.

4.7. Prediction Modes

The action selection method explained in Section 4.6.4 is based on training data for the self-motion predictor, obtained from a constrained DOF exploration phase, as well as training data for the feature predictor, obtained from observing the motion of external objects.
In this section, I introduce an additional method for action selection that learns from a previous interaction stage, in which the method of Section 4.6.4 is used to generate training data through the interaction experience.
In the following, the method of Section 4.6.4 is termed *first-stage mode* and the additional method that is going to be outlined is termed *second-stage mode*. Together with the con-

strained DOF exploration, the first-stage and second-stage modes form a learning method for sensory-motor sequences in multiple consecutive stages where one stage bootstraps sequences serving as training data for the subsequent stage. The benefit of introducing these multiple interaction stages is the robot's ability to self-generate training data in order to improve its skill performance over time. This principle is applied to the development of the reaching skill evaluated on the TOMM robot (Figure 33). The second-stage mode partly

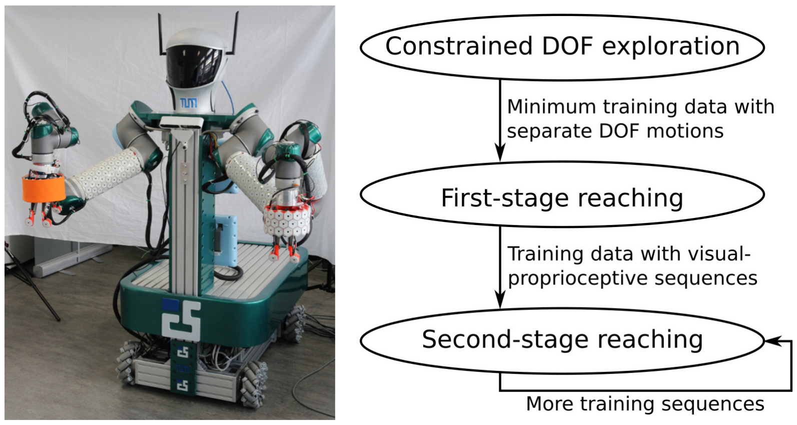

Constrained DOF exploration
Minimum training data with separate DOF motions
First-stage reaching
Training data with visual-proprioceptive sequences
Second-stage reaching
More training sequences

Figure 33 TOMM robot (on the left) and the stages of learning (on the right) for the development of reaching skill. The first-stage and the second-stage reaching are each realized by the first-stage and second-stage prediction mode, respectively. Note that the first-stage and second-stage mode are regarded as a general method for staged learning. A particular example is the reaching skill evaluated on the TOMM robot (on the left). Photo of the robot and illustration of the stages by Burger [34].

yields an improvement in terms of reaching times, i.e. a reduction of the time needed to reach a distal goal in the robot's workspace.

4.7.1. First-Stage Mode

Algorithm 15 shows the process of generating the sequences by the first-stage prediction. Algorithm 15 implements the method explained in Section 4.6.4. It evaluates each DOF separately to choose between the available motions. In order to select the best motion for a single DOF, it recalls the trained sequences resembling the three (c) available motions (*up*, *down*, *idle*) from the self-motion predictor. Then, it uses the resulting visuo-proprioceptive sequences to calculate the predicted relative feature changes $\Delta \mathbf{v}_s$ and $\Delta \mathbf{p}$. By comparing the predicted changes $\Delta \mathbf{v}_s$ with the visual goal $\mathbf{v}_g$ (i.e. through minimization of Function (47)), the algorithm selects the best suited motion for the current DOF and updates the relative feature changes (Equations (53) and (54)). It repeats this procedure for every DOF to be controlled. The outer for-loop appends each computed visuo-proprioceptive sample $(\mathbf{v}_f \quad \mathbf{p})$ to the generated sequence S_{gen}.
An illustration of the working principle of the first-stage mode for a simple 2 DOF example is shown in Figure 34(a).

Algorithm 15 Sensory-motor sequence generation using the *first-stage prediction mode*. Comments are in curly brackets. Within the outer for-loop, the computations implement the method in Section 4.6.4 and yield a new visuo-motor sample at each cycle. The outer for-loop generates the visuo-proprioceptive sequence by predicting the samples in advance through a closed-loop manner, in which each PAS output sample is fed back to its input, instead of immediately sending the motor part to the robot.

Requirements: SMP (Algorithm 11),
optionally FP (Algorithm 13) for prediction of occluded $\mathbf{v}_g$ or for prediction of occluded $\mathbf{v}_f$ in case of head control (for an example scenario of $\mathbf{v}_g$ and $\mathbf{v}_f$, see bottom of Figure 35)

Parameters: length L (integer) of closed-loop prediction,
prediction length factor κ (integer),
alteration parameters set to $a_f = -1$ and $a_s = -1$

Input: visual goal $\mathbf{v}_g$, visual feature $\mathbf{v}_f$, proprioceptive state $\mathbf{p}$

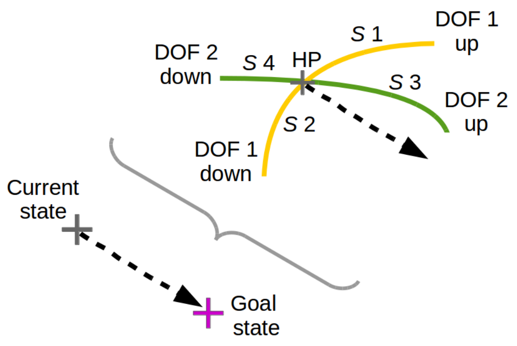
Output: generated visuo-proprioceptive sequence $S_{gen} = (\mathbf{s}_0, \mathbf{s}_1, ..., \mathbf{s}_{L-1})$ with $\mathbf{s}_i = (\mathbf{v}_f \quad \mathbf{p})_i$

```
 1: Initialize empty generated sequence $S_{gen} \leftarrow []$
 2: for $i \leftarrow 0; i < L; i \leftarrow i + 1$ do
 3:   for each DOF $d$ do
 4:     $e_v \leftarrow ||\mathbf{v}_g - \mathbf{v}_f||$   {Equation (49)}
 5:     $m \leftarrow \lfloor \kappa \cdot e_v \rfloor$   {Equation (50)}
 6:     for each learned motion $c \in$ [up, down, idle] do
 7:       $\Delta\mathbf{v}_s(c, m) \leftarrow \mathbf{v}_f(c, m) - \mathbf{v}_f(c, 0)$   {$\mathbf{v}_f(c, m)$ and $\mathbf{v}_f(c, 0)$ by SMP (Algorithm 11)}
 8:       $V(c, m) \leftarrow ||\mathbf{v}_g + a_f \cdot \mathbf{v}_f + a_s \cdot \Delta\mathbf{v}_s(c, m)||$   {Equation (47); if $\mathbf{v}_f$ represents the position
    of an external object *and* if it is not visible, then $\mathbf{v}_f := \mathbf{v}_f^{(pred)}$ by FP (Algorithm 13); the same
    applies to $\mathbf{v}_g$}
 9:       Store $c$ and $V(c, m)$
10:     end for
11:     $c_w \leftarrow c$ with minimum $V$   {Selection of the winning sequence $c_w$}
12:     $\Delta p_d(c_w, m) \leftarrow p_d(c_w, m) - p_d(c_w, 0)$   {Equation (52)}
13:     $p_d \leftarrow p_d + \Delta p_d(c_w, m)$   {Equation (53)}
14:     $\mathbf{v}_f \leftarrow \mathbf{v}_f + \Delta\mathbf{v}_s(c_w, m)$   {Equations (54) and (55)}
15:   end for
16:   $S_{gen} \leftarrow (S_{gen}, (\mathbf{v}_f \quad \mathbf{p}))$
17: end for
18: return $S_{gen}$
```

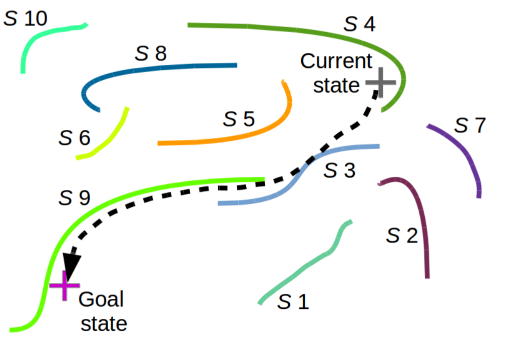
4.7.2. Second-Stage Mode

The first-stage mode generates sequences that are used as training data for the second-stage mode.

In contrast to the first-stage mode using the computed relative changes, the second-stage mode operates with the learned sequences and uses their absolute sample values. A comparative illustration of the working principle of the two prediction modes is shown in Figure 34. When the PAS computes a new output sample using the second-stage mode, every incoming



(a) First-stage prediction mode



(b) Second-stage prediction mode

Figure 34 Illustrated working principle of the first-stage prediction and second-stage prediction mode in the visual space. In the first-stage prediction mode (34(a)), the method generalizes from the memory formed by the constrained DOF exploration. In this example, the current state is moved along a trajectory (indicated by the black dashed line with arrow) to the goal state by reducing the position of DOF 1 and increasing the position of DOF 2. In the second-stage prediction mode (34(b)), the method generates the new sequence (black dashed trace) by exploiting the attractor dynamics formed by a previous interaction stage that used the first-stage mode to generate sequences to form the attractor landscape memorized by the EO-MTRNN of the SMP.

sample triggers the prediction of N_S different samples, one sample per learned sequence. The sequences have been generated and learned from a previous first-stage interaction in order to form an attractor landscape. Among the N_S samples, the method selects a winner sample that has the minimum Euclidean distance to the goal. The prediction of a sample, given an input sample and a particular sequence learned, is based on context recognition. Context recognition updates the context states of the EO-MTRNN of the SMP to correspond to an activation state of the IO group that is closest to the given input sample. Algorithm 16 details the working principle of the second-stage mode.

Algorithm 16 Sensory-motor sequence generation using the *second-stage prediction mode*. Comments are in curly brackets. The notation $\mathbf{v}_t$ refers to $\mathbf{v}_f$ (visual feature) at time t.

Requirements: SMP in second-stage (Algorithm 12)
Parameters: length L (integer) of closed-loop prediction,
number of learned sequences N_S (integer) generated in the first-stage,
maximum length R_{max} (integer) of mental rehearsal,
steps to goal δ (integer) with $\delta \leq L_{max}$,
one_step (boolean) {indicate whether prediction of $t+1$ or $t+\delta$}
Input: input sample $\mathbf{s} := \mathbf{s}_t = (\mathbf{v}_t \quad \mathbf{p}_t)$
Output: visuo-proprioceptive sequence S_{gen} with output samples $\mathbf{s}_{t+1+i}$ with $i \in \mathbb{N} \mid 0 \leq i < L_{max}$
Variable: $\mathbf{b}$ to temporarily store the samples

```
 1: Initialize empty generated sequence $S_{gen} \leftarrow$ []
 2: for $i \leftarrow 0; i < L; i \leftarrow i+1$ do
 3:   $\mathbf{b} \leftarrow \mathbf{s}$
 4:   for $c \leftarrow 0; c < N_S; c \leftarrow c+1$ do
 5:     for $j \leftarrow 0; j < R_{max}; j \leftarrow j+1$ do
 6:       $e_v[c][j] \leftarrow ||\mathbf{v}_g - \mathbf{v}_f(\mathbf{b}, c)||$   {$\mathbf{v}_f(\mathbf{b}, c)$ by SMP (Algorithm 12), i.e. comp. $\mathbf{v}_f$ given $\mathbf{b}$, $c$}
 7:       $\mathbf{b} \leftarrow (\mathbf{v}_f(\mathbf{b}, c) \quad \mathbf{p}(\mathbf{b}, c))$   {$\mathbf{p}(\mathbf{b}, c)$ by SMP (Algorithm 12), i.e. comp. $\mathbf{p}$ given $\mathbf{b}$, $c$}
 8:     end for
 9:   end for
10:   $(c_w, j_w) \leftarrow (c, j)$ with minimum $e_v$   {Selection of winning sequence $c_w$ and winning sample $j_w$}
11:   if one_step then
12:     $\mathbf{s}_{t+1}(\mathbf{s}, c_w)$ by SMP (Algorithm 12)   {i.e. compute $\mathbf{s}_{t+1}$ given $\mathbf{s}$, $c_w$}
13:     $S_{gen} \leftarrow (S_{gen}, \mathbf{s}_{t+1})$
14:   else
15:     for $j \leftarrow 0; j < \delta; j \leftarrow j+1$ do
16:       $\mathbf{b}(\mathbf{s}, c_w)$ by SMP (Algorithm 12)   {i.e. compute $\mathbf{b}$ given $\mathbf{s}$, $c_w$}
17:       $\mathbf{s} \leftarrow \mathbf{b}$
18:     end for
19:     $S_{gen} \leftarrow (S_{gen}, \mathbf{s}_{t+\delta})$
20:   end if
21: end for
22: return $S_{gen}$
```

4.8. Experiments: PAS Learns to Coordinate the NAO Robot

On the NAO robot, I deployed a PAS for head control and another PAS for arm control. The working principle of each PAS (first-stage prediction mode) stays the same, no matter whether it learns head or arm coordination. The purpose of the experiments on the NAO robot is a thorough evaluation of the first-stage mode of the PAS and the resulting skills that emerge.
An example of the conducted experiments on the NAO robot is summarized by the following videos:
The video-link `https://youtu.be/1s1IlVbd444` is a media attachment to the paper [220]. In particular, this video shows the training of the FP (of the PAS) in order to learn the possible motions, i.e. trajectories, of a target object through observation.
The video-link `https://youtu.be/fVzIxPxT7MY` is a media attachment to the paper [221]. This video shows the autonomous learning of coordination of head and arm.

4.8.1. Stages of Learning and Operation

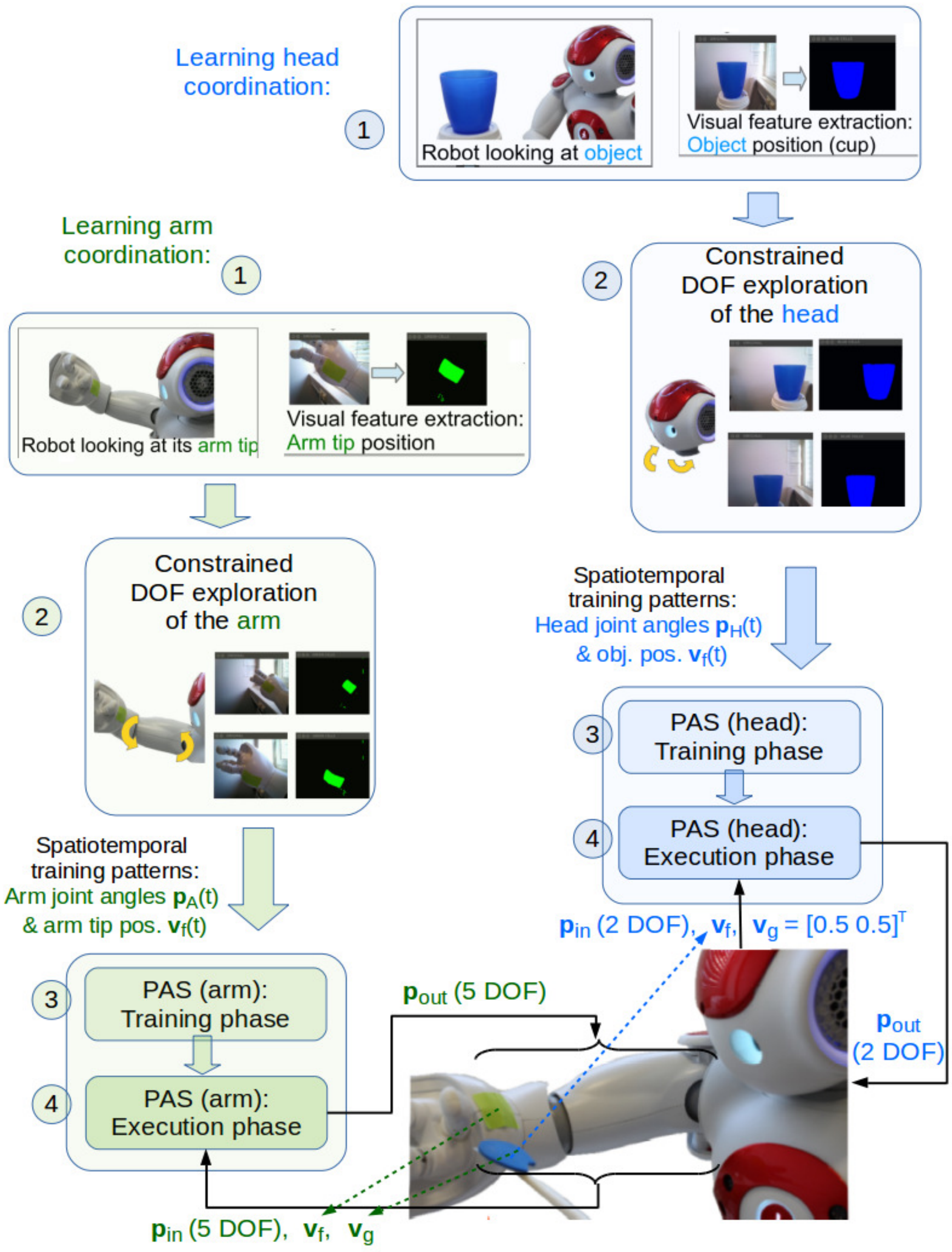
In general, when conducting experiments with the PAS, the detailed stages of learning and operation consist of:

1. Home positioning and feature extraction (Section 4.3)
2. Constrained DOF exploration (Section 4.4, Algorithm 6 also used for home positioning)
3. Training phase of the PAS (by using BPTT of the EO-MTRNN of Chapter 3)
4. Execution phase of the PAS that is subdivided into first-stage and second-stage prediction mode (Section 4.7)

Note that these stages of learning and operation are the same for any robot that is controlled by the PAS. Also note that due to its nature of sequence recall, the second-stage prediction mode makes only sense for the emergence of an improved arm coordination, e.g. for improving the reaching skill.
In case of the NAO robot, Figure 35 gives an overview on these stages of learning and operation. It also shows the setup for the experiments conducted on the NAO robot.

4.8.2. Additional Training of the Feature Predictor

For a prediction of the motions of *external* objects, the feature predictor needs to be trained as well, with example sequences of object motion.
I trained the feature predictor with 46 sequences of the visual feature $\mathbf{v}_f$ that represented the position of an object (a cup). Each sequence contained about 30 to 60 samples. These training sequences consisted of various horizontal, vertical, and idle motions of the cup in the FOV of the robot. While the cup was moved in front of the robot, the robot did not move, also



Learning head coordination:
1
Robot looking at object
Visual feature extraction: Object position (cup)
2
Constrained DOF exploration of the head
Spatiotemporal training patterns: Head joint angles $\mathbf{p}_H(t)$ & obj. pos. $\mathbf{v}_f(t)$
3
PAS (head): Training phase
4
PAS (head): Execution phase
$\mathbf{p}_{in}$ (2 DOF), $\mathbf{v}_f$, $\mathbf{v}_g = [0.5\ 0.5]^T$
$\mathbf{p}_{out}$ (2 DOF)
Learning arm coordination:
1
Robot looking at its arm tip
Visual feature extraction: Arm tip position
2
Constrained DOF exploration of the arm
Spatiotemporal training patterns: Arm joint angles $\mathbf{p}_A(t)$ & arm tip pos. $\mathbf{v}_f(t)$
3
PAS (arm): Training phase
4
PAS (arm): Execution phase
$\mathbf{p}_{out}$ (5 DOF)
$\mathbf{p}_{in}$ (5 DOF), $\mathbf{v}_f$, $\mathbf{v}_g$

Figure 35 Stages of learning and operation. One PAS learns to coordinated the head, another PAS learns to coordinate the arm.

its head was in resting state, “observing” the object motions. During this phase, the cup has to be visible for the robot.

The EO-MTRNN of the feature predictor learned these sequences in order to compute *predictions* of $\mathbf{v}_f$, annotated as $\mathbf{v}_f^{(pred)}$ (Algorithm 13). This means that each time when the cup is occluded or covered, the feature $\mathbf{v}_f$ is replaced by $\mathbf{v}_f^{(pred)}$ in the first-stage mode of the

PAS (Line 8 of Algorithm 15).

4.8.3. EO-MTRNN Parameterization of PAS (NAO Robot)

Table 17 shows the parameterization of the EO-MTRNN (Chapter 3) of the PAS for the experiments on the NAO robot. In the following, SMP$_H$ denotes the self-motion predictor and FP$_H$ denotes the feature predictor, both of the PAS for *head* control. SMP$_A$ denotes the self-motion predictor of the PAS for *arm* control. The values for learning rates and momentum

EO-MTRNN hyperparameter	N_{IO}	N_{FC}	N_{SC}	τ_{IO}	τ_{FC}	τ_{SC}
SMP$_H$	4	5	5	7	20	20
FP$_H$	2	5	5	7	20	20
SMP$_A$	8	15	5	7	20	20

Table 17 EO-MTRNN hyperparameterization of the PAS for the experiments on the NAO robot.

were the same as in Table 11 of Chapter 3.
Note that the EO-MTRNN was used in a minimum configuration, i.e. relative few neurons, AHE was not used. Pre-processing was de-activated, i.e. each input-output dimension was directly mapped to an input-output neuron. Also, early stopping was de-activated.

4.9. Results: Emergence of Coordination Skill on the 2 DOF Head (NAO Robot)

I evaluated the emergent behaviour generated by the PAS that learned to control the 2 DOF head.
The following results were collected based on:

1. a constrained DOF exploration (Algorithm 6) of the 2 DOF *head/neck joint*,
2. the training of the feature predictor (Algorithm 13), and
3. the first-stage prediction mode (Algorithm 15).

Note that a constrained DOF exploration generating the training data for the self-motion predictor of the PAS is necessary to bootstrap the coordination skill, whereas a training of the feature predictor of the PAS is optional. Nevertheless, the additional training of the feature predictor improves the emergent skill, since the PAS becomes more robust against the temporal loss of visual features.
During the constrained DOF exploration of the head, the system collected 48 sample vectors. Note that this number can vary depending on the parameters set for the constrained DOF exploration (Algorithm 6). At least four samples have to be collected per recorded sequence

in order to yield the results shown in this section, in particular to let the coordination skill emerge. This is a minimum amount of 20 sample vectors (five sequences with four samples per sequence).
In this experiment, the third dimension v_z of the goal $\mathbf{v}_g$ or feature $\mathbf{v}_f$ is neglected, since it does not influence coordinated motions performed by the head/neck joint only. The goal feature was $\mathbf{v}_g = (0.5 \quad 0.5)^T$. This corresponds to the middle of the robot's FOV. The extracted object position of the blue cup (Figure 35) is represented by $\mathbf{v}_f = (x \quad y)^T$ in the normalized image plane of the NAO's top camera.
As a metric for the evaluation of the system's performance, I used the visual error given by

$$e_v = ||\mathbf{v}_g - \mathbf{v}_f^{(obs)}|| \tag{56}$$

where $\mathbf{v}_f^{(obs)}$ is the *observed* visual feature, i.e. the observed object position.
In order to evaluate the performance with fixed prediction lengths, Equation (50) was skipped in the process. The prediction length for the self-motion predictor of the first-stage mode was fixed to $m = 4$ for computing $\Delta \mathbf{v}_s$ and $m = 3$ steps for computing Δp.
Note that the PAS motor output was directly sent to the robot, i.e. the loop parameter was set $L = 1$ in Algorithm 15.
All together, these settings led to the emergence of two different meaningful behaviours: object tracking and object evasion. Here, evasion means to move the head such that the distance to a target point in the visual space, expressed by the visual error, is maximized, i.e. the object is brought out of view. Although this seems to be a rare behaviour, it may be beneficial when it is coupled with an additional mechanism like attention selection or inhibition of return. In that case, it yields behaviours like engaging or disengaging in attention, which can be integrated seamlessly into other behaviour patterns.

4.9.1. Switching between Behaviours through Alteration Parameters

The alteration parameters yielding these two types of behaviour are:

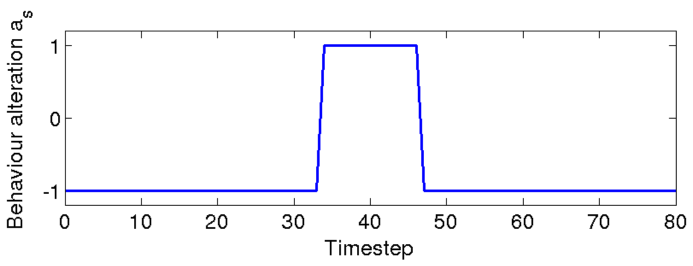
1. Object tracking (i.e. engaging in attention): $a_s = -1$, $a_f = -1$
2. Object evading (i.e. disengaging in attention): $a_s = +1$, $a_f = -1$

The change of the sign of one parameter (a_s) switches between these behaviours, given the same training data (constrained DOF exploration). This switching from tracking behaviour to evasive behaviour and vice versa is depicted in Figure 36.

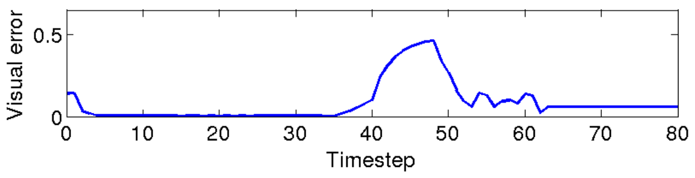
4.9.2. Tracking an Object and Predicting its Position

I evaluated the object tracking behaviour by moving an object (a cup) in front of the robot. The object was moved fast from one location to an other. The PAS enabled the robot to track the object by head motions or saccades[9]. During this interaction, the cup was always visible

[9] The NAO robot's cameras are fixed in its head. The actuation of the head/neck joint such that an object is brought into centre of view corresponds to a saccade.



(a) Behaviour alteration parameter a_s



(b) Visual error depending on a_s for switching between tracking and evading

Figure 36 Switching from tracking behaviour to evasive behaviour and vice versa, depending on the alteration parameter a_s. The beginning of interaction ($a_s = -1$) is characterised by a certain amount of visual error. The robot reduces this error by a saccade to the object. At time step 34, the parameter was changed to $a_s = +1$ (by tactile touch). Due to this change, the robot's behaviour switched from tracking to evasive motions, as reflected by the increasing visual error. At time step 47, the parameter was changed back to $a_s = -1$ and the behaviour switched from evasion to tracking, therefore reducing the visual error.

to the robot.

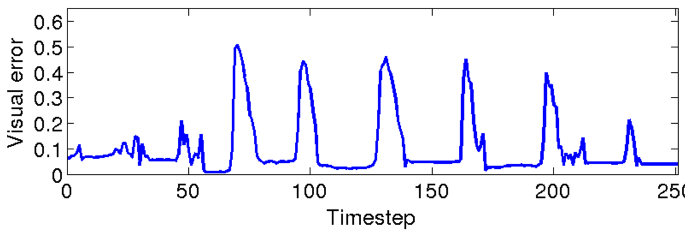
The parameters for the feature predictor were $R = 5$ and $P = 3$. The tracking results are shown in Figure 37. Since the object has been always visible during this interaction, the observed object position was used to compute the motor commands (Algorithm 15). Nevertheless, the feature predictor was additionally active in order to compare its signals with those observed (Figures 37(b) and 37(c)).

I evaluated the effect of the prediction length P of the feature predictor, independent of any robot motion. For this purpose, the focus was on the time steps 50 to 150 of the interaction. The observed object position was fed into the feature predictor. While keeping the recognition length $R = 5$ constant, the predictions were recorded for $P = 3$, $P = 5$, and $P = 15$, respectively. The results are shown in Figure 38.

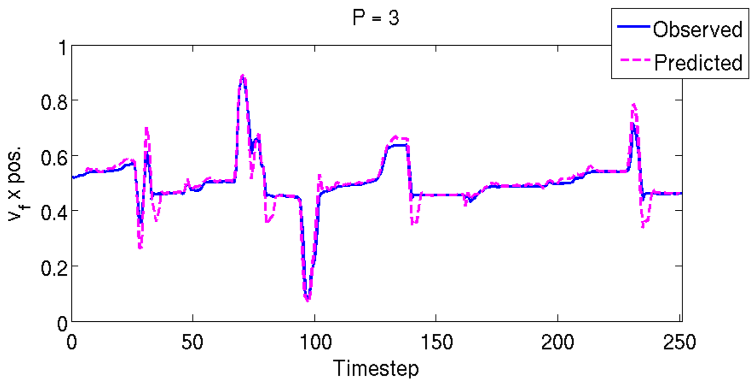
4.9.3. Coping with Temporal Loss of Feature

In the previous interaction scenario, the visual feature has been always observable. However, there are situations with temporal loss of the visual feature due to occlusion or fast motions. In those situations, traditional tracking mechanisms either stop moving entirely or search the entire visual FOV, taking time until the object is found again.

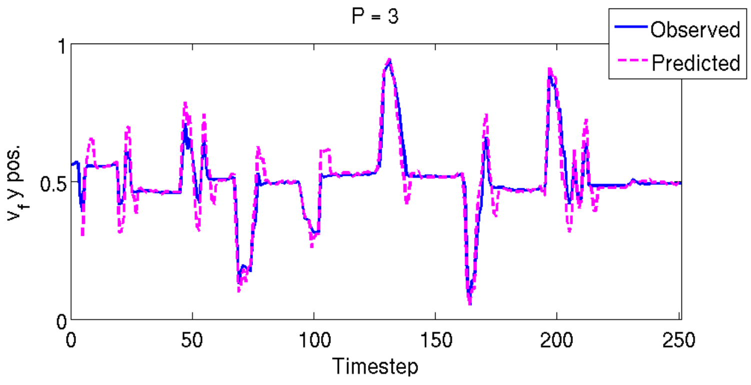
The PAS works with the predicted feature in case of feature loss (Line 8 of Algorithm 15). Figure 39 shows an example of the temporal loss of features that was caused by moving the object of interest behind an other object. In this experiment, the cup was moved behind a piece of paper for a relative short time (ca. 1 to 2 seconds), before it became visible again.



(a) Visual error

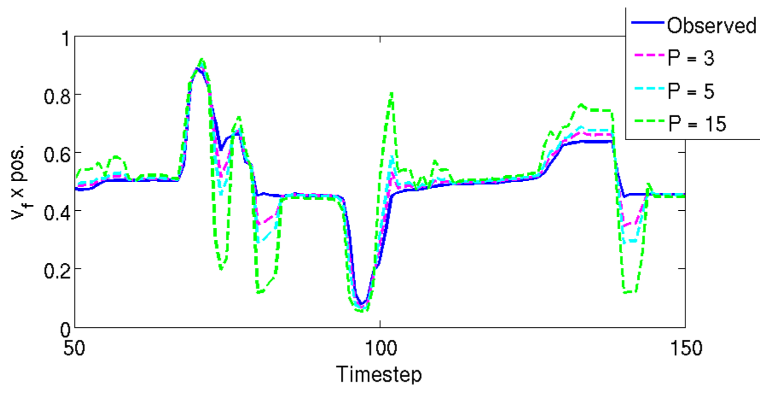


(b) Visual feature: x-position

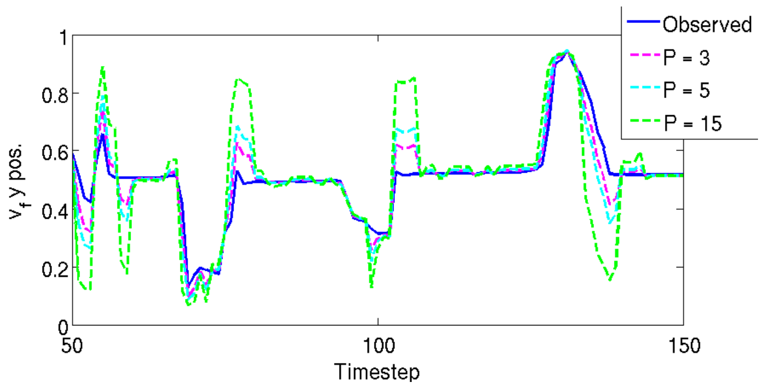


(c) Visual feature: y-position

Figure 37 Tracking a moving object of interest (a cup) by head/eye. Every peak in the visual error (37(a)) indicates a big and fast motion of the object in the robot's FOV. The robot reacts with a saccade in order to reduce the visual error. After a saccade, the remaining visual error is smaller than 4–5 %. The motion of the object is shown in 37(b) and 37(c), with x- and y-position in the robot's FOV. The magenta signal is the prediction by the feature predictor.

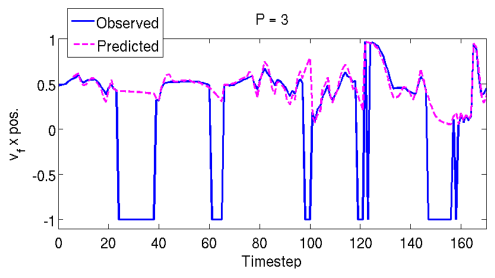


(a) Visual feature: x-position

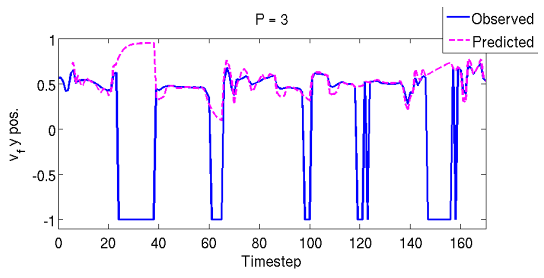


(b) Visual feature: y-position

Figure 38 Effects of the prediction length P on the feature prediction. The observed object position (blue) from time steps 50 to 150 of the previous interaction (Figure 37) was fed into the feature predictor. Different values of P lead to different predicted signals (dashed). The recognition length $R = 5$ was kept constant. The bigger the value for P, the more far ahead in time the prediction is.



(a) Visual feature: x-position



(b) Visual feature: y-position

Figure 39 Coping with temporal loss of visual feature representing the object position. The object of interest (cup) was moved behind an other object (a piece of paper) in front of the robot. The loss of the observed visual feature (i.e. object position) due to occlusion is represented by $\mathbf{v}_f^{(obs)} = (-1 \quad -1)^T$. In that case, the PAS relies on the predicted feature $\mathbf{v}_f^{(pred)}$ (magenta) computed by its feature predictor and continues to generate the appropriate motor commands in order to move the robot in an anticipatory manner. The parameters of the feature predictor were $R = 5$ and $P = 3$.

4.9.4. Tracking an Object with Adaptive Prediction Length

Instead of using fixed prediction lengths $m = 4$ and $m = 3$ for respectively computing $\Delta\mathbf{v}_s$ and $\Delta\mathbf{p}$ through the self-motion predictor of the PAS, Equation (50) was now used in the process. Equation (50) yields an adaptive prediction length depending on the observed visual error.

I conducted an additional object tracking experiment with the prediction length factor $\kappa = 15$ in order to evaluate whether an adaptive prediction length improves the tracking skill. The result in terms of the visual error is shown in Figure 40. Although this interaction contained

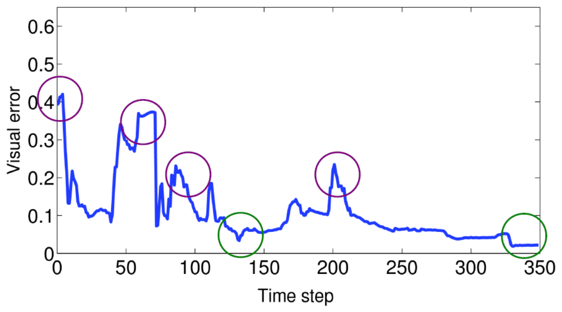


Figure 40 Visual error while the PAS is tracking an object (a cup) by head motions, with adaptive prediction length. At the onset of interaction (purple circle on the very left), the object appears in the corner of the robot's FOV causing a large error that the robot reduces immediately by orienting its camera towards the object. The remaining purple circles indicate occlusion (time steps 60 and 72) and sudden motions of the object (time steps 85 and 200). Once the object is only slightly moved or stationary (green circles), the remaining visual error is 1–2 %. This is an improved result compared to a fixed prediction length.

temporal occlusion or covering of the object, the robot's behaviour is still robust enough in terms of a continuous tracking that is made possible through the feature predictor.

4.10. Results: Emergence of Coordination Skill on the 5 DOF Arm (NAO Robot)

I evaluated the emergent behaviour generated by the PAS that learned to control the 5 DOF arm[10]. In order to evaluate the scalability of the PAS in terms of the number of DOF, I validated the learning of arm coordination when the robot's head was in a resting position oriented towards the arm.

The following results were collected based on:

1. a constrained DOF exploration (Algorithm 6) of the 5 DOF *arm*, and

2. the first-stage prediction mode (Algorithm 15).

During the constrained DOF exploration of the arm, the system collected 126 sample vectors. In case of arm control, the visual feature $\mathbf{v}_f$ represents the robot's arm/hand position that

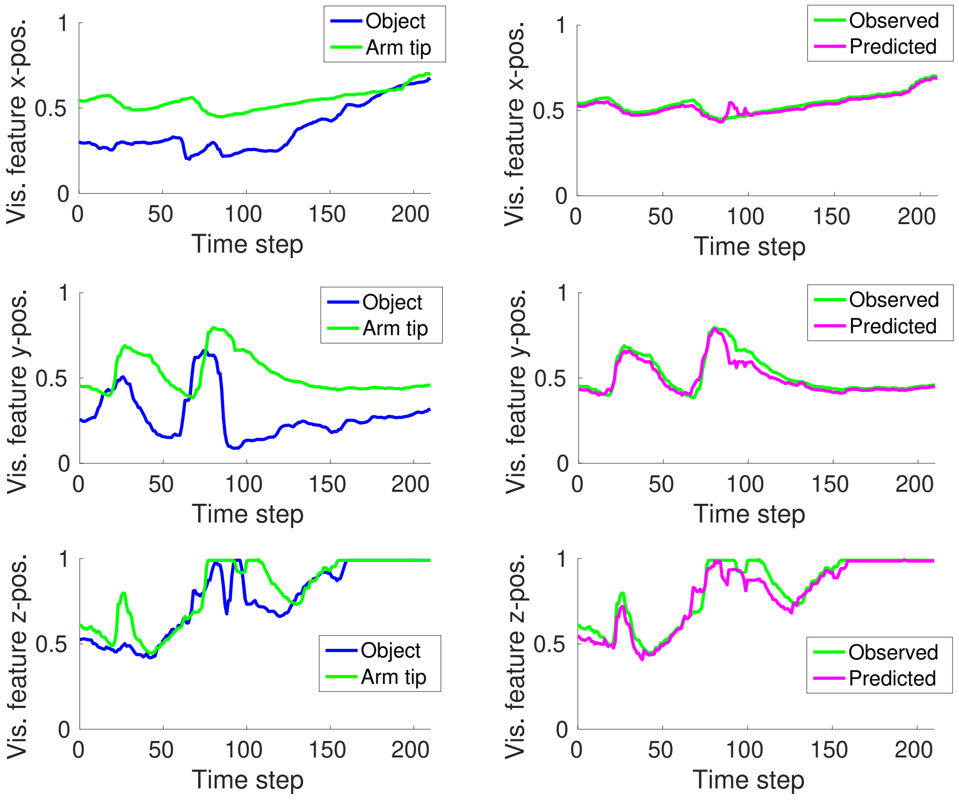
[10]On the NAO arm, the fingers are considered as DOF 6, but they are neglected for the scope of this work as mentioned earlier.

is always predicted by the PAS (Equations (54) and (55)). In this experiment, the feature predictor was not trained[11].
The visual goal pattern $\mathbf{v}_g$ was set to be the object position. Here, the object of interest was the blue marker on the chopstick, see Figure 35.
In Algorithm 15, the alteration parameters were $a_s = -1$ and $a_f = -1$. The prediction length factor was $\kappa = 8$. Note that the PAS motor output was directly sent to the robot, i.e. the loop parameter was set $L = 1$.

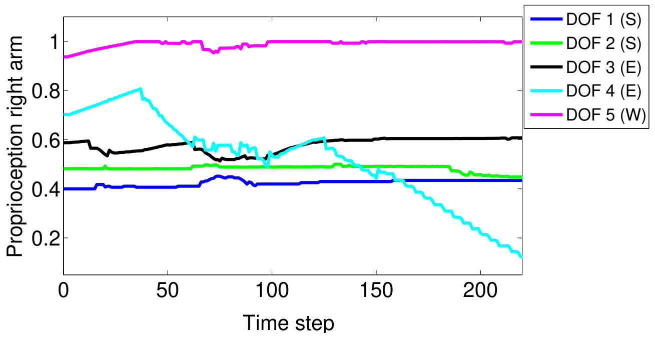
4.10.1. Reaching for an Object

The object was moved in a random pattern in front of the robot's FOV and the PAS controlled the arm to reach for the object. The results of the interaction are shown in Figure 41. The right side of Figure 41(a) shows the prediction of the arm tip computed by the self-motion predictor of the PAS. The best prediction was 150 milliseconds ahead in time, the sampling rate was 20 Hz.

[11]The feature predictor needs only be trained for the prediction of *external* object trajectories, i.e. for motions that are independent from the robot's own motion. Optionally, it could be trained to predict the motions of the target $\mathbf{v}_g$.



(a) Observed target and arm tip (left side), and prediction of arm tip (right side)



(b) Motor signals corresponding to 41(a). DOF 3 and 4 (elbow) were mostly active, regulating the distance to the object.

Figure 41 Observed and predicted visual features during the reaching (conducted on the NAO robot). The left side of 41(a) shows the observed target object (blue marker on a chopstick end, represented by the blue signal) that was moved randomly the robot's FOV, with variations in the distance. The arm tip (green signals) was approaching the object. The robot reached the object around time step 160. The remaining error (0.2 in y-position) is due to the colour markers of arm tip and target, and means that the object is within the robot's grasp. The right side of 41(a) shows the observed arm tip feature (green) and its prediction (magenta) during the reaching. The PAS could accurately predict the observed arm tip 150 milliseconds ahead (sampling rate was 20 Hz). The corresponding motor signals are shown in 41(b).

4.11. Experiments: PAS Learns to Coordinate the TOMM Robot

In addition to the NAO robot, the PAS was also validated on the Tactile Omnidirectional Mobile Manipulator (TOMM) robot [52]. The purpose of the conducted experiments of the PAS on TOMM is:

- the validation of the scalability in terms of DOF, i.e. from 5 DOF to 6 DOF,
- the validation of operation on a robot with an entire different morphology than the NAO, i.e. different motors, greater physical size and greater workspace, and
- the validation of the multi-stage developmental learning for reaching (see Figure 33).

Note that the stages of learning and operation are the same as described in Section 4.8.1. The focus here is the comparison between the performance of first-stage and second-stage mode applied to the multi-staged learning of the reaching skill.
Note that in contrast to the NAO robot, the TOMM robot's head is fixed and cannot move. Nevertheless, TOMM's head is equipped with a stereo vision system, its visual field is significantly larger, and its workspace is larger as well.
For the evaluation of the reaching, goal positions have been defined in the workspace of the robot. The goals are distributed in a grid-like pattern across the visual space. The robot attempts to reach each goal from the same home position of its arm.
Similarly to the NAO robot's hand, a colour marker was attached to TOMM robot's end-effector in order to yield the 3D position $\mathbf{v}_f$ of its end-effector in the robot's FOV. During the experiments, a goal position $\mathbf{v}_g$ was considered as reached if the Euclidean distance between $\mathbf{v}_g$ and $\mathbf{v}_f$ is less than 0.05 in the (normalised) visual space. Physically, this corresponds to approximately 35 mm in the workspace.
For the stages of reaching, closed-loop prediction of the PAS was applied. This means that $L > 1$ in Algorithms 15 and 16. The upper limit for the length of sequence prediction was $L = 10$.
For the closed-loop prediction, the prediction error e_p was introduced that is the Euclidean distance between the observed and predicted position of the end-effector:

$$e_p := ||\mathbf{v}_f^{(obs)} - \mathbf{v}_f^{(pred)}|| \tag{57}$$

An additional condition to the predictions by Algorithms 15 and 16 was that a sequence prediction terminated if $e_p > 0.15$.
The experiments on the TOMM robot were jointly conducted with Burger in [34].
An example of the conducted experiments on the TOMM robot is summarized by the following video: `https://youtu.be/okZz9HPRrZI` that is a media attachment to the paper [34]. The video shows the multi-staged reaching on TOMM.

4.11.1. EO-MTRNN Parameterization of PAS (TOMM Robot)

Table 18 shows the parameterization of the EO-MTRNN (Chapter 3) of the PAS for the experiments on the TOMM robot. Since only one PAS was used that controlled the arm given static goals, only the self-motion predictor of the PAS needed to be parameterized, i.e. SMP$_A$. The following parameters show the configuration of the SMP$_A$, the configuration varied dependent on the training data used for the first-stage and second-stage reaching. The values for

EO-MTRNN hyperparameter (SMP$_A$)	N_{goals}	N_{IO}	N_{FC}	N_{SC}	τ_{IO}	τ_{FC}	τ_{SC}
first-stage	378	9	10	10	7	20	20
first-stage	27	9	10	10	5	10	100
second-stage	27	9	10	10	5	10	100

Table 18 EO-MTRNN hyperparameterization of the PAS for the experiments on the TOMM robot.

learning rates and momentum were the same as in Table 11 of Chapter 3.
The EO-MTRNN was used in a minimum configuration, i.e. relative few neurons, AHE was not used. Pre-processing was de-activated, i.e. each input-output dimension was directly mapped to an input-output neuron. Also, early stopping was de-activated.

4.12. Results: Emergence of Coordination Skill on the 6 DOF Arm (TOMM Robot)

The following results were obtained based on:

1. constrained DOF exploration (Algorithm 6) of the 6 *DOF arm*,
2. the first-stage prediction mode (Algorithm 15), and
3. the second-stage prediction mode (Algorithm 16) with fallback to first-stage if reaching time exceeds 60 seconds.

The results of the constrained DOF exploration are shown in previous Section 4.4.4. The exploration yielded 13 sequences with up to 29 samples each. The SMP of the PAS was trained with these sequences.

4.12.1. First-Stage Reaching

The first-stage prediction mode yielded the emergence of a first-stage reaching that was evaluated with 378 goal positions distributed in the robot's FOV. This evaluation particularly considers the reaching time for each goal. The result is a visual reachability map, where the colours encode the reaching time, shown in Figure 42. In total, 66 % of the 378 specified goals were reached within less than 180 seconds. The average reaching time of the successfully

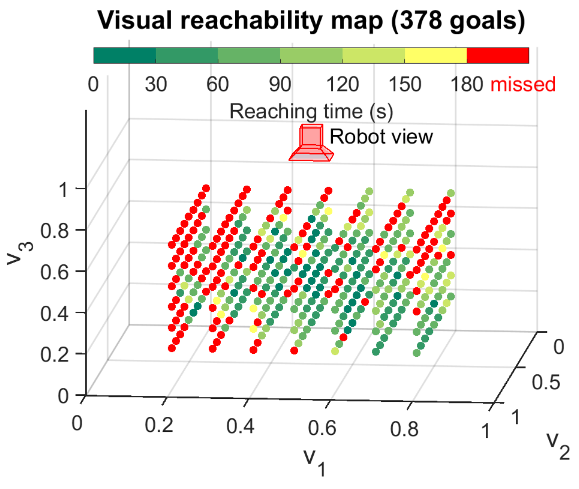


Figure 42 Visual reachability map obtained as a result of first-stage reaching of 378 goals; reaching conducted on the TOMM robot. The axes represent the normalized 3D visual space. The colour of the goals represent the reaching time (in seconds), as indicated by the colour bar. Goals in red were not reached within 180 seconds. Goals in other colours were reached in less than 180 seconds. After each attempt to reach a goal (successful or not), the end-effector returned to the same home position. The results were visualized by Burger in [34].

reached goals was 72 seconds.

4.12.2. Second-Stage Reaching

An important step of the developmental process (right side of Figure 33) is the usage of the sensory-motor sequences generated in the first-stage as training data for the PAS in second-stage prediction mode.

In order to obtain a comparative evaluation, a test set of 27 goals was used for the first-stage. The first-stage PAS was trained again using 13 sequences obtained through constrained DOF exploration. The second-stage PAS was trained with 57 sequences containing a total of 449 samples that were observed and recorded during the first-stage reaching of various goals. Figure 43 shows the result of these two stages.

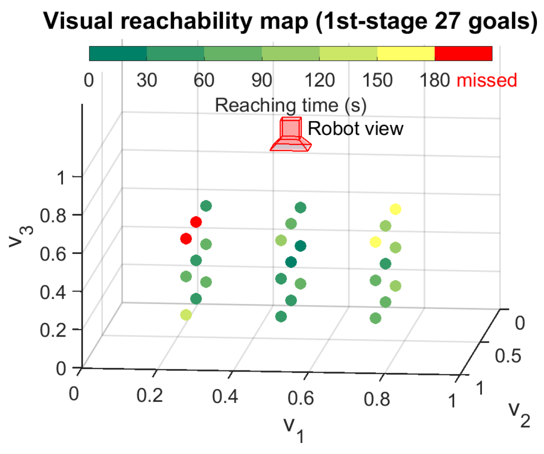
Corresponding to Figure 43, the reaching time for each of the 27 goals for both first-stage and second-stage reaching is listed in Table 19. In both experiments (Figure 43(a) and Figure 43(b)), 92 % of the target goals were reached after an average of approximately 77 seconds.

An improvement in the reaching skill was accomplished by the second-stage prediction mode (Figure 43(b)) that was able to reach 9 goals before falling back to the first-stage mode, i.e. it reached those goals in less than 60 seconds. This reduced their average reaching time by 21 %. Moreover, the reaching times of other goals were reduced by up to 59 % through the second-stage prediction mode and consecutive fallback to first-stage. Also, one goal was reached that was missed before in the first-stage.

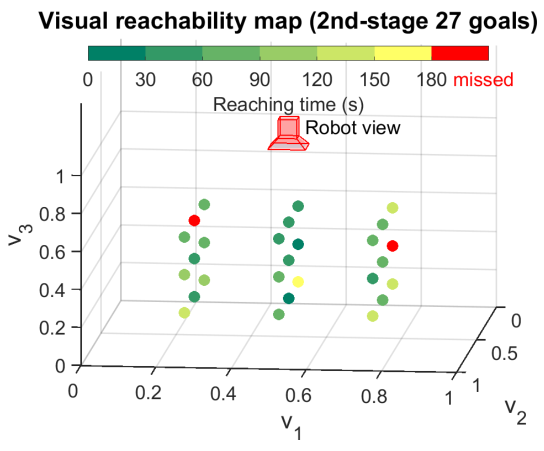
A limitation was that the second-stage prediction mode appeared to have insufficient training data to generate appropriate sequences for some of the goals. For 10 goals, the reaching time increased significantly (greater than 15 seconds) and one goal that was reached in pure first-stage mode could not be reached anymore.

Visual dimension			**1st-stage reaching** (PAS trained with 13 sequences)	**2nd-stage reaching** (2nd-stage PAS trained with 57 sequences)
v_{g1}	v_{g2}	v_{g3}	reaching time t_{reach} (in seconds)	reaching time t_{reach} (in seconds)
0.25	0.25	0.20	89.33	106.05
0.25	0.25	0.40	60.95	67.13
0.25	0.25	0.60	34.63	71.82
0.25	0.50	0.20	55.12	43.11
0.25	0.50	0.40	38.47	34.27
0.25	0.50	0.60	missed	missed
0.25	0.75	0.20	147.74	137.46
0.25	0.75	0.40	81.73	98.23
0.25	0.75	0.60	missed	68.27
0.50	0.25	0.20	63.79	152.17
0.50	0.25	0.40	13.50	14.67
0.50	0.25	0.60	59.48	58.65
0.50	0.50	0.20	40.86	29.90
0.50	0.50	0.40	11.62	43.12
0.50	0.50	0.60	64.96	32.08
0.50	0.75	0.20	56.74	88.16
0.50	0.75	0.40	51.44	80.75
0.50	0.75	0.60	104.66	59.85
0.75	0.25	0.20	92.73	134.86
0.75	0.25	0.40	104.76	missed
0.75	0.25	0.60	170.09	123.51
0.75	0.50	0.20	72.68	73.49
0.75	0.50	0.40	44.03	89.49
0.75	0.50	0.60	119.85	65.00
0.75	0.75	0.20	88.44	124.28
0.75	0.75	0.40	84.62	56.05
0.75	0.75	0.60	176.07	72.18
mean(t_{reach})			77.13	76.98
$N_{reached}/N_{goals}$			92.59 %	92.59 %

Table 19 Reaching time for a test set of 27 goals corresponding to Figure 43. The reaching phases were conducted with first-stage mode and second-stage mode of the PAS. The second-stage mode used fallback to first-stage if the reaching exceeded 60 seconds. The missed goals are indicated in red and the shortest reaching times are indicated in green. The times were recorded by Burger in [34]. Note that the average reaching time was reduced by 21 % for 33 % of the goals, i.e. for one third of the robot's workspace (goals evenly distributed).



(a) First-stage reaching



(b) Second-stage reaching

Figure 43 Visual reachability map obtained as a result of first-stage reaching (43(a)) followed by second-stage reaching (43(b)) of 27 goals of a test set; reaching conducted on the TOMM robot. The colour of the goals represent the reaching time (in seconds), as indicated by the colour bar. After each attempt to reach a goal (successful or not), the end-effector returned to the same home position. When operated in second-stage mode, the PAS fell back in first-stage mode if the reaching time exceeded 60 seconds. The results were visualized by Burger in [34].

4.13. Discussion

In the following, I discuss the obtained results based on the priority table of the scientific goals (Table 16).

4.13.1. Learning of Hand-Eye Coordination: Achievements, Limitation, Network Configurations, Training Time and Skill Execution Time

4.13.1.1 Overall Achievements

The proposed PAS successfully learned basic hand-eye coordination in a progressive manner, i.e. it learned to coordinate the head/eye on a robot with moving head (NAO) and then it learned to coordinate its arm. This order can also be swapped, since one PAS was deployed for head control and another PAS for arm control, independent of each other. The PAS for head control (denoted by PAS$_H$) and the PAS for arm control (denoted by PAS$_A$) were evaluated separately from each other, with only one active PAS at a time in order to make firm statements about its performance in learning to coordinate the particular limb. Nevertheless, both PAS (head/eye and arm) can be also active simultaneously.
On the NAO robot, the learned skill of object tracking yielded an accuracy of 1–2 % in the remaining visual error. The early reaching (first-stage) yielded accurate predictions of the robot's arm tip of up to 150 milliseconds ahead in time at a sampling rate of 20 Hz. For the PAS Algorithm 15, an adaptive prediction length is preferable over a fixed one, since it yields more fine-tuned motions depending on the observed error.
The learning was also possible on an entirely different robot (TOMM). This substantiates that the PAS works on multiple robot platforms. The obtained skills in terms of meaningful robot behaviour were object tracking and evading by head/eye motion (that can be used to engage and/or disengage in attention), and an early reaching by the arm, reaching that can be refined or improved through a subsequent interaction stage (second-stage).
In addition, learning to coordinate head and arm was realized in an autonomous manner, i.e. through a constrained DOF exploration that generated the very first training data, instead of providing them by a human. Through a first interaction stage that already yields a meaningful skill executed with the arm (e.g. reaching), the PAS can generate further data and use them for a subsequent interaction stage to refine the skill.

4.13.1.2 Limitation

From a developmental point of view, hand-eye coordination is acquired by first learning head-vision coordination (i.e. the causal relation between the head movement and the resulting change of visual image) and then followed by learning arm-vision coordination (i.e. the causal relation between the arm movement and the resulting change of visual position of the arm tip). PAS$_H$ learns head-vision coordination and PAS$_A$ learns arm-vision coordination. One may argue that if hand-eye coordination is really acquired, the robot should be able to reach objects *regardless* of head position. However, this particular ability requires an additional memory component for the extracted visual features. In the current setting, the PAS$_A$ is dependent on the head position, since PAS$_A$ receives the visual features from the camera of the robot's head. In order to be able to reach for objects regardless of head position, a kind of short-term buffer memory is required that provides the visual features for the PAS$_A$. When the PAS$_H$ turns the head in other directions away from the target object, the buffer memory provides the image at the time when the target object was still in the FOV to the PAS$_A$. Thus, by the inclusion of a short-term memory, the history of target visual features can be captured and the skill of reaching an object regardless of head position can be realized. This also justifies the need for a cognitive architecture that uses the PAS modules as key building blocks and complements them by providing a short-term memory for the visual and motor features.

4.13.1.3 Configurations of the EO-MTRNN, Training Time, and Execution Time of Constrained DOF Exploration

At its core, the PAS uses the EO-MTRNN (Chapter 3) in order to learn and to rehearse sensory-motor sequences. In all the experiments conducted with the PAS, a minimum parameterization effort was made. Pre- and postprocessing was not used, each dimension of the sensory-motor data was directly mapped to an input-output neuron. Also, early stopping was not used due to the small amount of training data. Thus, in accordance with the notation in Chapter 3, the EO-MTRNN blocks of the PAS were in the configuration preprocessing *off* and early stopping *off*. Furthermore, the autonomous estimation of network hyperparameters (by the evolutionary optimizer) was not used. An autonomous estimation of hyperparameters would rather be beneficial if training data would accumulate for the second-stage reaching and thus the learning error of the network should be kept low. Sections 4.8.3 and 4.11.1 provide the number of neurons and their corresponding timescales. The values for learning rates and momentum were the same as in Table 11 of Chapter 3.
The proposed constrained DOF exploration collected a minimum amount of samples for training the PAS. The length of the sequences collected through constrained DOF exploration was short, i.e. ranging from 4 to 12 samples (per sequence) for the NAO robot, and to 29 samples (per sequence) for the TOMM robot. For the former case on the NAO robot, the EO-MTRNN blocks of the PAS was used in a single timescale mode for the context neurons (same value for τ_{FC} and τ_{SC} in Table 17). The choice of the single timescale mode can be justified by the small amount of training data containing short non-overlapping sequences. For the latter case on the TOMM robot, a multiple timescale mode (i.e. different value for τ_{FC} and τ_{SC}, respectively, in Table 18) was chosen, since the second-stage mode was trained with more sequences (i.e. 57 sequences, with a total of 449 samples, obtained from the first-stage interaction). The multiple timescale mode reduces the error in learning a larger dataset, especially when the sequences are longer.
Although the PAS algorithms require spatiotemporal learning, this does not imply that the proposed version of the MTRNN is the only type of recurrent neural network suitable for the PAS. In theory, the MTRNN can be replaced by a different type of neural network. Important is that

the network facilitates learning of spatiotemporal patterns and their representation. Recurrent networks facilitate this requirement, making them the best choice.
The time needed for the execution of a constrained DOF exploration is relatively short and depends on the robot and the number of DOF that are explored. For example, it takes approximately 20 seconds for the arm (5 DOF) of the NAO robot, for its head (2 DOF) it takes only a few seconds. On the TOMM robot, it takes longer (roughly one to a few minutes) for the arm (6 DOF). This also depends on the (low-level) industrial controller of the motors.
Also, the time needed to train the EO-MTRNN blocks of the PAS was relatively short. For example, given the data obtained through constrained DOF exploration, it took approximately 5 minutes to train the self-motion predictor of the PAS to control the arm of the NAO robot. Another example is the time for both constrained DOF exploration and training of the EO-MTRNN of the PAS to control the 6 DOF arm of the TOMM robot. There, the constrained DOF exploration and the consecutive EO-MTRNN training (with the parameters shown in Table 18) took only a total of 10 minutes.

4.13.1.4 Execution Time of the Reaching Skill across Multiple Stages

An objection would be the reaching times obtained during the evaluation of the first-stage and second-stage mode (Section 4.12). In average, the time that the TOMM robot needed to reach a particular goal in its workspace was approximately 76 to 77 seconds. Note that this reaching time is primarily caused by a slow velocity set for the (low-level) industrial controller of the TOMM robot. A slow velocity was set because of security reasons and to avoid problems with the industrial controller. Note that the velocity was pre-set and constant, since the PAS processes positions as the only physical quantity[12]. For these reasons, the obtained reaching times are significantly higher than the necessary computation time of the PAS algorithms.
A quantitative improvement was the reaching time for some of the goals in the robot's workspace. Table 19 reveals that the average reaching time was reduced by 21 % for 33 % of the goals. Since the goals were evenly distributed in the robot's workspace, this yields a reduction of the reaching time by 21 % for one third of the workspace.

4.13.2. Learning the Dynamics of External Entities

The feature predictor of the PAS can learn the dynamics of an external entity (i.e. motions of an object) that is moving in the FOV of the robot and is relevant for interaction. In this work, the only feature was position in the robot's visual field. The EO-MTRNN was used to recognize the observed trajectory and to predict future positions. The performance of the feature predictor mainly depends on the amount of training sequences. In the experiments of head/eye coordination, the predictor was trained with 46 sequences representing possible motions of the object that the robot may encounter during the interaction. The feature pre-

[12]The EO-MTRNN of the PAS is *not* restricted to positions, it can be any other physical quantity such as velocity or force. However, since the poverty of stimulus was modelled, the aim was to cope with the absolute minimum of physical quantity.

dictor (Algorithm 13) requires the setting of two parameters, the recognition parameter R and the prediction parameter P. Parameter R depends on the length of the learned sequences. The longer a learned sequence is, the greater the value of R that should be chosen to yield a correct recognition. The parameter P determines the number of samples that should be predicted in advance. For small values of R, small values of P should be chosen to yield a prediction close to ground truth.
In the conducted experiment where a cup was moved in the robot's FOV, $P = 3$ yielded acceptable results (Figure 38), although the training data represented simple, mostly linear, motions of the cup and the buffer size for recognition was relative small ($R = 5$). Prediction performance can be improved by including longer sequences with more complex motions, e.g. curved trajectories, into the training data.
A thorough analysis of the parameter space was out of scope and is potential future work.
It is worth noting that due to the usage of a network based on predictive coding, the predicted features do not have to represent positions only. For instance, a feature vector might also contain other physical quantities such as velocities or torques, dependent on the application target. Nevertheless, in this work, the PAS was designed to concur with the developmental point of view that includes principles of skill emergence based on minimum information, e.g. minimum training data, minimum physical quantities, etc. For the robot platform, a small velocity was pre-set and kept constant, and position (in visual and motor space) was the only physical quantity upon which the skills emerged.

4.13.3. Robustness to Temporal Loss of Sensory Data

The experiments showed that both self-motion predictor and feature predictor contribute to the robustness of the PAS in cases of temporal loss of sensory data during interaction. In the experiment of the emergent reaching (Figure 41(a)), the self-motion predictor accurately predicted the arm tip position approximately 150 milliseconds ahead at 20 Hz. Given the same sampling rate, the feature predictor yielded predictions of external object positions at 150 milliseconds as well, determined by the inverse of sampling rate times number of steps predicted ahead (i.e. 50 milliseconds times 3).
The results of the conducted experiments suggest that the two predictor blocks of the PAS yield short-term feature prediction, i.e. predictions ahead in time ranging from 50 to 450 milliseconds, depending on the sampling rate. In case of the feature predictor, the prediction results are highly dependent on the quality of training data.
The generation of motor commands was never interrupted in cases of feature loss. The PAS Algorithms 15 and 16 generated motor commands in every computation cycle, independent of whether the visual feature was observable (i.e. visible in the camera) or not. This helped the robot to move in a predictive manner in case of occlusions or coverings of the visual feature.

4.13.4. Integration of Action Selection and Action Generation into One Framework

The PAS Algorithms 15 and 16 integrate action selection and action generation on the sensory-motor level into one computational framework. In the first-stage mode (illustrated by Figure 34(a)), the action selection evaluates predicted sensory outcomes for (primitive) chunks of DOF motion. The action is generated through assembly of the separate chunks. In the second-stage mode (illustrated by Figure 34(b)), the selection process evaluates sequences that emerged through the previous stages, i.e. constrained DOF exploration and subsequent first-stage selection and generation.
A benefit of this developmental scheme is that primitive, simple actions can be generated in a particular stage during development, and then be re-used by the same MTRNN-based framework in a next stage later (although the prediction mode is different). In fact, the selection process itself generated meaningful actions that in turn were used for selection in the subsequent stage in order to let more complex actions emerge, e.g. longer reaching movements. This can be seen as part of a hierarchy of selection and generation that is in accordance with recent findings in the frontal cortical lobes [14], [16].

4.14. Summary

In this chapter, I proposed the PAS as a new action selection system that contributes towards overcoming the problem of poverty of stimulus by using a relative simple interface (Section 4.3), by autonomously generating a minimum amount of training data (Section 4.4), and by using that data to bootstrap fundamental sensory-motor coordination skills.
To this end, I built on the predictive coding-based method that I proposed in Chapter 3 and extended it by biologically-inspired algorithms. This resulted into a proposed computational model that integrates action selection and action generation on sensory-motor level across multiple stages (Sections 4.6 and 4.7).
The conducted experiments yielded major highlights of the proposed PAS model, which are summarized by the following points:

- The PAS uses positions in the visual space and in the motor space as the only physical quantity. Target positions in motor space are sent to the robot's controller generating the motor torques.

- The PAS bootstraps sensory-motor coordination autonomously and with a minimum amount of training data, which is in accordance with the poverty of stimulus. In the experiments for instance, 48 samples were enough to bootstrap eye/head coordination yielding object tracking and evading, which can be re-used for engaging and disengaging in attention. The accuracy of object tracking was 1–2 % in the remaining visual error. An early reaching skill was bootstrapped from 126 samples. As part of the arm coordination, internal prediction of the arm tip was performed 150 milliseconds ahead in time at 20 Hz.

- The PAS yields an improvement in the coordination of the arm through multiple interaction stage (first-stage and second-stage). For 33 % of evenly distributed goals in the workspace, the second-stage reaching yielded a reduction of the average reaching time by 21 %.

- A relative simple configuration of the proposed EO-MTRNN (Chapter 3) is enough for bootstrapping sensory-motor coordination, i.e. no preprocessing, no early stopping, and non-optimized (i.e. without AHE). Note that the EO-MTRNN is always trained with multiple sensory-motor sequences simultaneously. The PAS switches between chunks of these sequences in a predictive manner in order to generalize to meaningful behaviour.

- The PAS is robust to short-term losses of sensory features. The PAS can predict visual features of the robot's body (e.g. arm tip) and of external objects. Features were predicted up to 3 steps ahead. Depending on the sampling rate (7–20 Hz), this yields predictions ranging from up to 150–450 milliseconds ahead in time.

- The PAS works on different robot platforms, each with different number of DOF. The PAS system was successfully validated on two robots, NAO and TOMM. The number of DOF that the system learned to coordinate ranged from 2 to 6. Although the validation was done with robot limbs up to 6 DOF, the PAS does not seem to be limited to that number.

Table 20 provides a comparative summary on the proposed PAS. It shows the types of PAS (i.e. first-stage, second-stage), their deployment, the resulting sensory-motor skills, and the key quantitative results obtained.
Although the PAS can cope with short-term[13] loss of sensory features, the predictions generated for a long-term[14] feature loss (e.g. due to occlusions, coverings) would be unreliable. A desirable cognitive capability would be to generate a forward model based on a previous interaction experience, a model that supports the PAS to facilitate a robust behaviour even for long-term feature loss.
In this chapter, each type of PAS was evaluated separately. Now, a cognitive architecture is desirable that integrates several PAS (and an additional type of predictor) into one framework, and provides a short-term memory for the visual and motor features. The cognitive architecture that builds on the PAS is presented in the next chapter.

[13]approximately 500 milliseconds
[14]ranging from several seconds or minutes to hours

Type of PAS	**Limb controlled**	**Origin and size of training data**	**Sensory-motor skill or capability**	**Quantitative results**
PAS$_H$	head (2 DOF)	C-DOF exploration, 48 samples	object tracking, object evading	tracking accuracy of 1–2 % (remaining visual error)
PAS$_A$	arm (5–6 DOF)	C-DOF exploration, 126 samples	(early) reaching	prediction of self feature (arm tip) up to 150 ms ahead in time (20 Hz)
PAS$_A^{(2)}$	arm (5–6 DOF)	reaching experience, 449 samples	faster reaching	for 33 % of the goals, the average reaching time was reduced by 21 %
PAS$_H$, PAS$_A$, PAS$_A^{(2)}$	head, arm	DOF exploration, interaction experience	Prediction of sensory-motor features	Prediction up to 150–450 ms ahead

Table 20 Different types of the proposed PAS, resulting sensory-motor skills, and quantitative results. PAS$_H$, PAS$_A$, PAS$_A^{(2)}$ denotes the PAS used for head control, arm control, and second-stage arm control, respectively.

5. Chapter

A Self-Verifying Cognitive Architecture

Building on my proposed PAS of the previous chapter, I propose a new developmental cognitive architecture — the SVCA — for a robust bootstrapping of coordination skills in the sensory-motor domain.

In this chapter, I describe the proposed cognitive architecture that is the main contribution of this thesis. This chapter has the following structure: Section 5.1 explains the motivation for a new cognitive architecture and lists the scientific goals. Section 5.2 describes my key idea that leads to a new developmental principle. The section also provides a conceptual overview on my proposed developmental model realized by the architecture. Section 5.3 outlines the design approach that was taken in order to develop the architecture. Section 5.4 presents the system overview. In particular, it presents how different PAS modules are integrated into a larger framework forming the architecture. Section 5.5 briefly explains the experimental setup that was used to validate the proposed architecture.

The experimental validation of the SVCA was done on the NAO robot, version 5, body type H25 [173]. Besides the NAO robot, the PAS that is one of the key components of the SVCA was also validated on the TOMM robot [52] in the previous chapter.

Section 5.6 presents the results of the conducted experiments. Section 5.7 discusses the obtained results. Finally, Section 5.8 summarizes this chapter.

In the following, the term *skill* refers to a *sensory-motor skill*, i.e. a robot's capability or behaviour in the sensory-motor domain (see also the definition of *skill and capability* in Section 2.7.1.3).

Note that I published most parts of this chapter in [222]. A minor part was published in [219].

5.1. Motivation for a New Cognitive Architecture

The proposed cognitive architecture is inspired by humans' ability to develop sensory-motor coordination skills early in their first months of life. While some skills are innate and already present at birth, for example reflex activity and limited sensory responses, a rich manifold of sensory-motor capabilities is acquired autonomously through exploration during the first year of life. From a developmental perspective, the autonomous acquisition of sensory-motor skills is of key importance [8], [211], [158]. It is important to point out that without the fundamental skills on the sensory-motor level, higher level cognitive skills would be hindered in their development [108], [172], [138].

The proposed cognitive architecture can be seen as a technical framework that models and artificially reproduces (parts of) the aforementioned capabilities at a basic sensory-motor level. At the same time, the goal of the proposed architecture is to overcome the limitations of existing architectures by supporting all criteria shown in Tables 3 and Table 4 of Chapter 2. Directed to the autonomous acquisition of sensory-motor skills, the goals of the architecture are:

- **Bootstrapping** of (initial) skills: The architecture should autonomously generate early hand-eye coordination from a minimum amount of sensory-motor data.

- **Continuous acquisition** of skills: The architecture should support a continuous skill acquisition by providing the necessary methods to capture new skills.

- **Incorporation** of new skills: The architecture should provide mechanisms that incorporate new skills into the repertoire of existing skills. Prior to this, a method should determine that a new skill encoded by sensory-motor sequences is stable.

- **Self-Verification** of signal or data: The architecture should have at least one metric that it can verify on its own, i.e. without human supervision, in order to determine skill stability and incorporation.

- **Improvement** of skill: The architecture should be able to improve in a particular skill. Improvement may encompass a faster skill execution time as well as increased robustness to environmental disturbances.

- **Adaptation to robot**: The architecture should *not* be tailored to a specific robot platform. Instead, it should provide an adaptation method that makes it possible to run the architecture on different robots with different joint encoders or motors, i.e. accurate types as well as inaccurate types with backlash.

5.2. Idea and Developmental Model

The key question is how a robot can autonomously acquire and develop meaningful behaviour stage by stage.

5.2.1. Meaningful Behaviour

I define behaviour as meaningful if it serves a particular goal in a certain context, e.g. goal-directed actions, or if it makes sense for a human observer interacting with the robot [41]. I also consider a behaviour as meaningful if it serves as a basis for the emergence or development of a related subsequent behaviour. For instance, an object tracking behaviour is meaningful [220], since it enables the agent to bring an object of interest into its FOV, thus creating the start conditions for a subsequent reaching behaviour [172].

5.2.2. Self-Verifying Multi-Stage Bootstrapping through Loops of Imaginary Trial and Physical Trial

My idea is that meaningful behaviour is bootstrapped through multiple stages where each stage generates training data to construct various predictors that are then used in a subsequent stage. The autonomous generation of training data is realized through processes of imagination (mental simulation) followed by physical trials with self-verification. The self-verification compares the sensory predictions of an imagined trial, for example an imagined reaching action, with the sensory observations of a physical trial.

Based on this idea, I propose the key principle of *self-verifying multi-stage bootstrapping through LITAPT*.

I implemented my proposed principle in a developmental cognitive architecture and validated it on the humanoid robot NAO. On a conceptual level, the development can be visualized by the timeline diagram shown in Figure 44. Figure 44 illustrates that the proposed developmen-

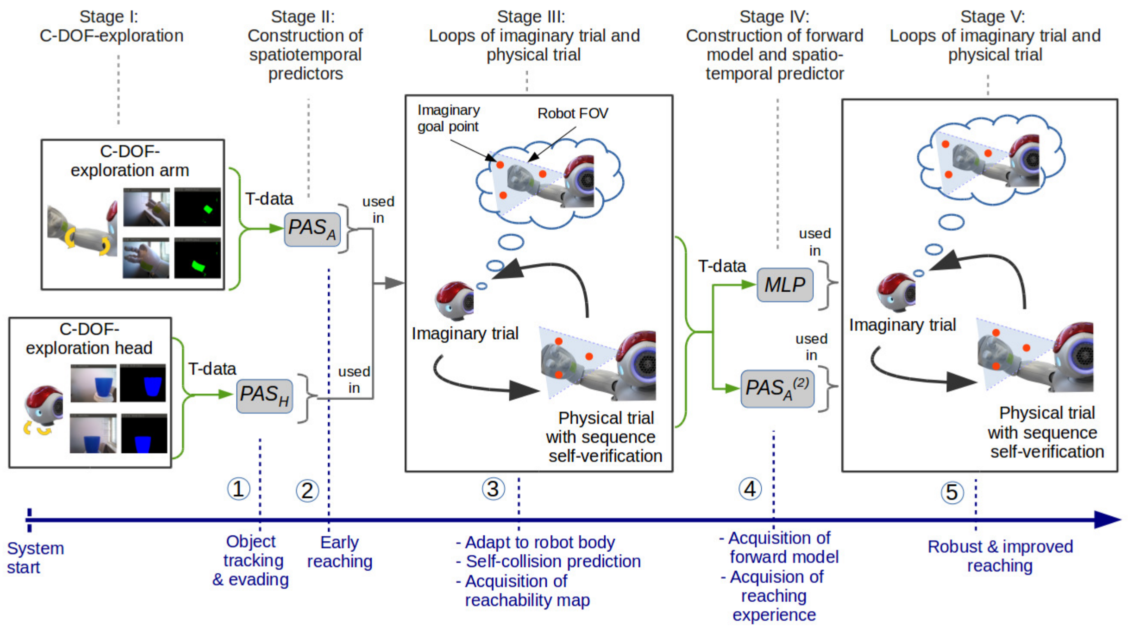

Stage I:
C-DOF-exploration
Stage II:
Construction of spatiotemporal predictors
Stage III:
Loops of imaginary trial and physical trial
Stage IV:
Construction of forward model and spatio-temporal predictor
Stage V:
Loops of imaginary trial and physical trial
C-DOF-exploration arm
C-DOF-exploration head
T-data
PAS_A
PAS_H
used in
Imaginary goal point
Robot FOV
Imaginary trial
Physical trial with sequence self-verification
MLP
PAS_A^(2)
1
2
3
4
5
System start
Object tracking & evading
Early reaching
- Adapt to robot body
- Self-collision prediction
- Acquisition of reachability map
- Acquisition of forward model
- Acquision of reaching experience
Robust & improved reaching

Figure 44 Timeline diagram of my proposed developmental model. The key principle is a self-verifying multi-stage bootstrapping process that operates with LITAPT for multiple purposes: adaptation to the robot platform through determining the optimal proprioceptive feedback, acquisition of an internal representation of predictor performance, generation of a forward model, and acquisition of sensory-motor experience in order to improve a skill.

tal model contains several developmental stages. In the first interaction stage, the constrained DOF exploration generates a minimum amount of sensory-motor data that is used to train two PAS modules (indicated by the grey boxes PAS). PAS$_H$ learns to control the head an yields tracking an external object as well as evading it. Object tracking and evasion can be used for engaging and disengaging in attention, respectively. PAS$_A$ learns to control the arm and yields an early reaching behaviour. In the next interaction stage, the robot performs loops of mental simulation and physical trial with self-verification. Within these imaginary and physical trials, the robot reaches for imaginary goal positions in its FOV.

The purpose of the LITAPT is manifold:

- Finding the optimal proprioceptive feedback: The architecture adapts itself to the robot body in terms of determining the optimal proprioceptive feedback and avoiding self-collision. Here, the proprioceptive feedback is the necessary difference between the consecutive target motor positions that cause a change in joint input in order to compensate for backlash in the joints or motors.

- Acquisition of an internal representation of predictor performance: The architecture acquires a reachability map that is evaluated through a confidence score representing the current predictor performance.

- Generation of a forward model: The architecture self-verifies the observed sensory outcome during every reaching attempt and uses the self-verified data to train a MLP. The trained MLP implements a forward model and supports the PAS. When both the PAS and the MLP are active, they enable the robot to perform robust reaching under disturbances, for example when its camera is occluded or its arm tip is entirely covered.

- Acquisition of sensory-motor experience: The architecture can use the self-verified data to train a second-stage PAS for arm control. The second-stage PAS improves the reaching performance by reducing the reaching time to some goals in the workspace (see the results in Section 4.12.2).

Note that the first-stage PAS modules are also used in stage V, together with MLP and second-stage PAS.

5.3. Architecture Design Approach

The proposed principle of self-verifying multi-stage bootstrapping can be decomposed into smaller state-of-the-art principles explained in Chapter 2. The aim is a technical framework that self-generates training data in order to construct various predictors through several developmental stages. The predictors are embedded into loops of imaginary trial and physical trial. A verification component analyses the discrepancy between the imagined and the real (i.e. physical) trial.

Besides the predictive coding approach [192], the architecture design was inspired by the principle of verification [180], [10], [185], [186] and the principle of grounding [180]. I drew further inspiration from developmental cognitive neuroscience [78] as well as the theory that the brain operates with signals encoding predictions and prediction errors [59], [60], [170]. In this case, the predictions are visuo-motor sequences computed by the PAS.

One of the implications of embodiment [138] would be the adaptation of the architecture to the robot's body. Ideally, this adaptation should be established by tuning very few parameters.

In sum, the proposed cognitive architecture blends together the following set of design principles: prediction and uniformity (through neural network-based modules such as PAS), bootstrapping, mental simulations, and self-verification.

Note that the cognitive capabilities and robot skills were not foreseen when building the architecture. The cognitive capabilities and robot skills are a result of the architecture design. The architecture starts the bootstrapping process from scratch, i.e. it has no *a priori* knowledge of either forward or inverse kinematics. It does not contain any model of the robot.
The architecture contains logic modules working on a symbolic rule-based level as well as predictor modules working with neural networks on a sub-symbolic level. Thus, my proposed architecture is a hybrid architecture according to the categorization in [210] and [208].

5.4. System Overview

This section is guided by the functional diagram of my cognitive architecture in Figure 45. The

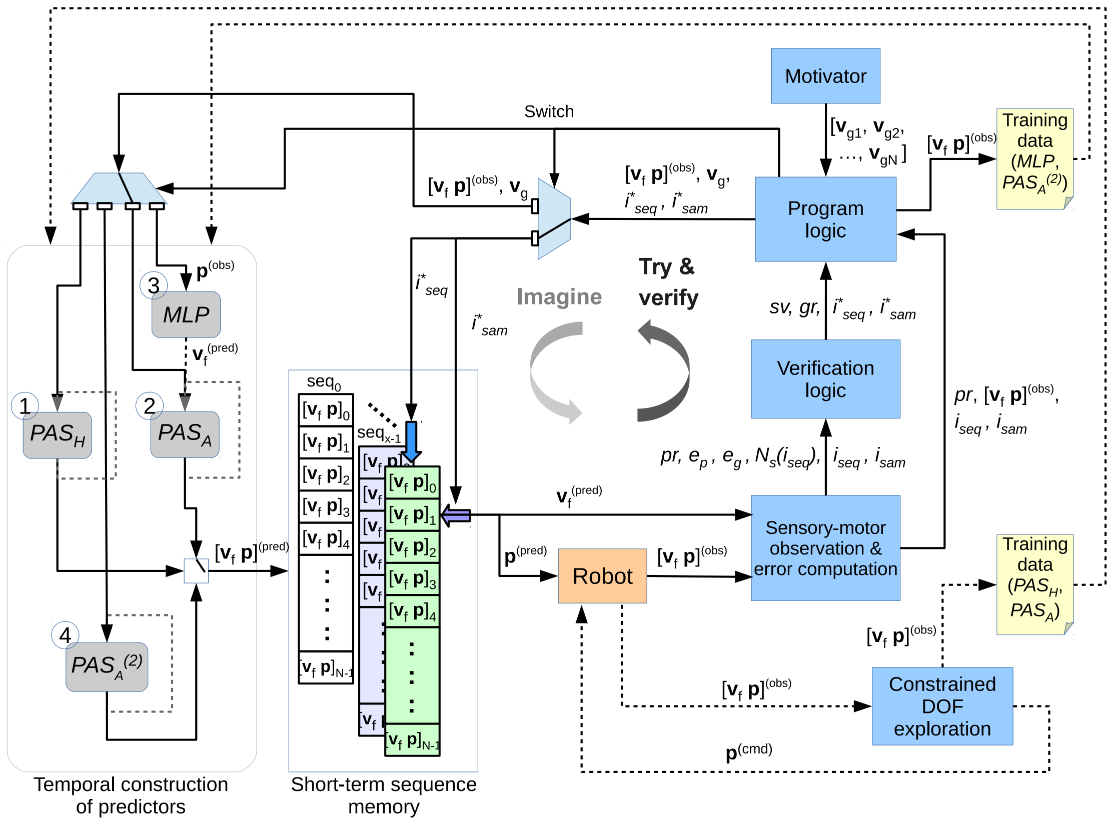

Motivator
Switch
Program logic
Verification logic
Sensory-motor observation & error computation
Robot
Constrained DOF exploration
Training data (MLP, PAS_A^(2))
Training data (PAS_H, PAS_A)
MLP
PAS_H
PAS_A
PAS_A^(2)
Imagine
Try & verify
Temporal construction of predictors
Short-term sequence memory
seq_0
seq_x-1
p^(obs)
v_f^(pred)
[v_f p]^(pred)
[v_f p]^(obs), v_g
p^(pred)
p^(cmd)
sv, gr, i*_seq, i*_sam
pr, e_p, e_g, N_s(i_seq), i_seq, i_sam
pr, [v_f p]^(obs), i_seq, i_sam

Figure 45 Functional diagram of my proposed cognitive architecture. The loops of imaginary and physical trial are indicated at the centre. The light grey curved arrow (labelled “Imagine”) indicates the process of imagination initiated by the program logic. The imaginary sensory-motor sequences are generated through the PAS modules in closed-loop mode and temporarily stored in the short-term sequence memory. Then they are executed by the robot and verified by the rule-based modules. This is indicated by the dark grey arrow (labelled “Try & verify”).

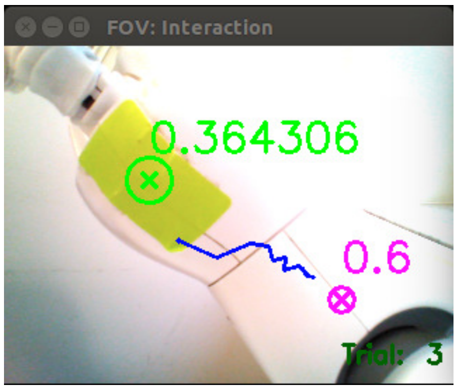
functional diagram can roughly be divided into three main types of components:

- Predictor pool (sub-symbolic level): Neural network-based predictors that are PAS$_H$, PAS$_A$, PAS$_A^{(2)}$, and MLP.

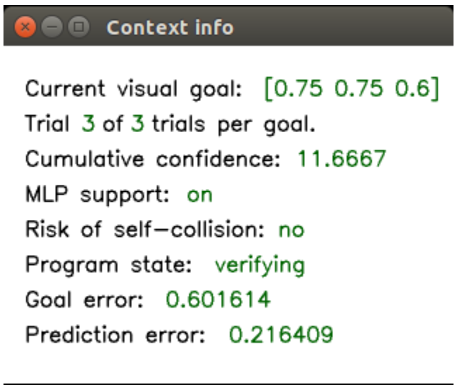
- Short-term sequence memory: A simple structure buffering the predicted sensory-motor sequences. This complements the PAS modules in order to overcome the limited coordination ability (described in Section 4.13.1.2 of the previous chapter).

- Rule-based modules (symbolic level): Program logic, verification logic, sensory-motor observation & error computation, motivator, and constrained DOF exploration.

Referring to my idea of self-verifying multi-stage bootstrapping (Section 5.2.2 and Figure 44), the proposed architecture realizes the loops of imaginary trial and physical trial. A snapshot of running the imaginary and physical trial is shown by Figure 46 that shows the interaction with an imaginary goal $\mathbf{v}_g$ (magenta), observed arm tip $\mathbf{v}_f^{(obs)}$ (green), and the imagined sequence (blue) consisting of predicted arm tip samples $\mathbf{v}_f^{(pred)}$ in the robot's FOV.



(a) Robot's FOV with key data displayed: Imaginary goal (magenta), observed arm tip (green), and the imagined sequence (blue) of predicted arm tip samples.



```
Current visual goal:  [0.75 0.75 0.6]
Trial 3 of 3 trials per goal.
Cumulative confidence:  11.6667
MLP support:  on
Risk of self-collision:  no
Program state:   verifying
Goal error:  0.601614
Prediction error:  0.216409
```

(b) Architecture state corresponding to 46(a).

Figure 46 Imaginary trial and physical trial from the robot's perspective (46(a)), and the corresponding architecture state (46(b)). In 46(a), the architecture first imagines a sensory-motor sequence (blue) consisting of predicted arm tip samples in the robot's FOV, aimed at the imaginary goal (magenta, at distance 0.6), before actuating the robot to bring the observed arm tip (green, at distance 0.36) to that goal.

5.4.1. Functional Interaction of Components

In Figure 45, the functional interaction of components can be summarized as follows: The constrained DOF exploration module generates sensory-motor data needed for constructing the predictor for head control (PAS$_H$) as well as for constructing the predictor for arm control (PAS$_A$). The constrained DOF exploration is connected to the system by dashed lines indicating that it is only needed for the very first bootstrapping stage and not used afterwards. The motivator module provides the goals in the visual space. The sensory-motor observation and error computation module delivers the current sensory-motor state and error signals to the verification logic. The verification logic feeds the verification results into the program logic that sends predicted motor samples from the short-term sequence memory to the robot. This short-term sequence memory is addressed by the sequence index i^*_{seq} (thick blue arrow) and the sample index i^*_{sam} (thick violet arrow), both computed by the program logic. The program logic also sends currently observed samples to the predictor pool. The training of various predictors happens gradually during the mental development of sensory-motor coordination (see Figure 44). The PAS modules realize imagination capability by their closed-loop circuitry (indicated by dashed grey lines). The program logic switches between the predictors and predictor pathways to determine the output of the predictor pool. The predictor pool output, that consists of visuo-motor sequences, is fed into the short-term sequence memory.

5.4.2. Data Types

Table 21 summarizes the data types exchanged between the modules of the architecture. Following the notation in Chapter 4, the vector $\mathbf{v}_f$ denotes the 3D position of a visual feature

Variable	**Type**	**Description**
$\mathbf{v}_g$	vector of floats	visual goal pattern that is a 3D goal position in the FOV
$\mathbf{x}_i = [\mathbf{v}_f \quad \mathbf{p}]$	vector of floats	visuo-motor sample, can be either the observed $\mathbf{x}_i^{(obs)}$ or the predicted $\mathbf{x}_i^{(pred)}$
e_p	float	prediction error, defined as the Euclidean distance between $\mathbf{v}_f^{(obs)}$ and $\mathbf{v}_f^{(pred)}$
e_g	float	goal error, defined as the Euclidean distance between $\mathbf{v}_f^{(obs)}$ and $\mathbf{v}_g$
i_{seq}, i_{seq}^*	integer (≥ 0)	current index, next index (of a sequence)
i_{sam}, i_{sam}^*	integer (≥ 0)	current index, next index (of a sample)
N_S	integer (≥ 0)	number of samples of a sequence
pr	boolean	motor *pattern reached*?
sv	boolean	predicted visuo-motor *sample valid*?
gr	boolean	*goal reached*?

Table 21 Data types in the architecture. The variable pr indicates whether a sent proprioceptive pattern has been reached or not. The variable sv indicates whether a predicted visuo-proprioceptive pattern has been found to be valid or not. The variable gr indicates whether a selected goal in the robot's FOV has been reached or not.

in the robot's FOV. It can be either an external feature (e.g. an object) or part of the robot's body, the arm tip or hand for instance. The vector $\mathbf{p}$ denotes the proprioceptive pattern that contains the DOF positions of a particular robot limb, e.g. the head or an arm.
Skill development has a particular focus on the arm coordination. In the following, $\mathbf{v}_f$ represents the position of the end-effector (arm tip marker) and $\mathbf{p}$ represents the DOF positions of the robot's arm.
The vector $\mathbf{x}_i$ denotes the visuo-proprioceptive sample, also referred to as visuo-motor sample, with i denoting the time index. At time t, a PAS is fed with either an observed sample $\mathbf{x}_i^{(obs)}$ or with a predicted sample $\mathbf{x}_i^{(pred)}$ (closed-loop). Given the input at t, a PAS outputs a predicted sample $\mathbf{x}_i^{(pred)}$ at the next time step $t+1$; the sample $\mathbf{x}_i^{(pred)}$ is put into the short-term sequence memory.
A sequence S of length N consists of the aforementioned visuo-motor samples, thus a sequence is defined by $S := (\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{N-1})$. Each sample $\mathbf{x}_i$ is a predicted sample that

is computed by a PAS.
The short-term sequence memory is simply a buffer for a given set of predicted sensory-motor sequences. Within the set, each sequence is directed to a particular goal in the robot's FOV.

5.4.3. Constrained DOF Exploration

This module has been explained in detail in Section 4.4. Here, a short summary of its operating principle suffices.
This method generates a minimum amount of training data and has been applied successfully on different robot configurations, such as NAO in [220] and [221], and TOMM in [34]. The purpose of the constrained DOF exploration is to generate the very first sensory-motor data that are used to bootstrap meaningful robot behaviour through the predictors. This exploration module contains a proportional control (P-control) for moving each DOF of a limb from its current angular position h into a target position $h \pm l$. Thus, each DOF can be moved to a given upper limit or a lower limit of exploration. Because l is very small, e.g. 5 % of the total range of a DOF, it is a *constrained* exploration of DOF. The motion commands are sent step-wise. In each step, only a small fraction of the given range of exploration is added or subtracted to the current position until the upper or lower limit is reached. Sensory-motor samples are recorded in each of these steps while moving from the home position to either the upper or lower limit. This constrained exploration of robot DOF only requires the arm to be positioned such that the arm tip (end-effector) is approximately in the middle of the robot's FOV. Because of the small range explored, the arm tip stays within the robot's FOV.

5.4.4. Predictive Action Selector

The PAS has been introduced and validated in Chapter 4. The PAS is the *spatiotemporal* predictor within the architecture framework[1]. It predicts the visuo-motor sequence S that is temporarily stored in the short-term sequence memory. Two prediction modes exist, depending on the training data and the controlled limb (Section 4.7). PAS$_H$ and PAS$_A$ always operate in the first-stage mode, since they are trained from data obtained through the constrained DOF exploration yielding only a minimum amount of data. PAS$_A^{(2)}$ is the second-stage predictor for arm control, it always operates in the second-stage mode. It is trained from data obtained through a previous interaction phase yielding goal-directed sequences (that encode a reaching skill, for instance).

5.4.5. Forward Model

A forward model is implemented by a MLP containing one hidden layer. In the architecture, the purpose of the forward model is to compensate for incomplete sensory-motor data. When the robot cannot perceive its arm tip or end-effector, for instance due to occlusion (i.e. covering), then the forward model predicts the arm tip location based on the current proprioceptive vector as its input. The forward model draws its training data from a previous interaction phase with goal-directed sequences. I utilize the MSE as error signal for training the MLP by

[1] The forward model by the MLP is a *spatial* predictor.

using backpropagation [149].

5.4.6. Sensory-motor Observation and Error Computation

This module checks whether a commanded motor vector $\mathbf{p}^{(pred)}$ has been reached, sends that commanded motor vector to the robot if not reached yet, and updates the Boolean flag pr accordingly. It also computes the prediction error e_p and the goal error e_g. The prediction error is the Euclidean distance between $\mathbf{v}_f^{(obs)}$ and $\mathbf{v}_f^{(pred)}$. The goal error is the Euclidean distance between $\mathbf{v}_f^{(obs)}$ and $\mathbf{v}_g$. Further outputs of this module are the number of samples N_S of a selected sequence[2], the sequence index i_{seq}, and the sample index i_{sam}. The indexes are updated by $i_{seq} \leftarrow i^*_{seq}$ and $i_{sam} \leftarrow i^*_{sam}$, after sensing the motor part of sample i^*_{sam} of sequence i^*_{seq} to the robot.
This module is realized by Algorithms 17 and 18.

Algorithm 17 Sensory-motor observation and error computation.

```
Input: $[\mathbf{v}_f\ \mathbf{p}]^{(obs)}$ from robot,
       $[\mathbf{v}_f\ \mathbf{p}]^{(pred)}$, $N_S$, $i^*_{seq}$, $i^*_{sam}$ from short-term sequence memory,
       $\mathbf{v}_g$ from motivator
Output: $\{pr, e_p, e_g, N_S(i_{seq}), i_{seq}, i_{sam}, [\mathbf{v}_f\ \mathbf{p}]^{(obs)}\}$

1: $pr \leftarrow$ updateMotorPattern($\mathbf{p}^{(obs)}$, $\mathbf{p}^{(pred)}$)
2: $e_p \leftarrow \|\mathbf{v}_f^{(obs)} - \mathbf{v}_f^{(pred)}\|$
3: $e_g \leftarrow \|\mathbf{v}_f^{(obs)} - \mathbf{v}_g\|$
4: $N_S \leftarrow$ get the number of samples of $i^*_{seq}$ from short-term sequence memory
5: $i_{seq} \leftarrow i^*_{seq}$
6: $i_{sam} \leftarrow i^*_{sam}$
7: return $\{pr, e_p, e_g, N_S(i_{seq}), i_{seq}, i_{sam}, [\mathbf{v}_f\ \mathbf{p}]^{(obs)}\}$
```

Algorithm 18 Function *updateMotorPattern(*$\mathbf{p}^{(obs)}$*,* $\mathbf{p}^{(pred)}$*)* as part of Algorithm 17.

```
Input: $\mathbf{p}^{(obs)} = [p_0\ p_1\ ...\ p_{N_D-1}]^{(obs)}$,
       $\mathbf{p}^{(pred)} = [p_0\ p_1\ ...\ p_{N_D-1}]^{(pred)}$
Output: motor pattern flag $pr$
Parameters: number of DOF $N_D$, position difference $\epsilon$

Ensure: Robot reaches given $\mathbf{p}^{(pred)}$
 1: $n \leftarrow 0$
 2: for $j \leftarrow 0; j < N_D; j \leftarrow j + 1$ do
 3:     if abs($p_j^{(obs)} - p_j^{(pred)}$) $< \epsilon$ then
 4:         $n \leftarrow n + 1$
 5:     end if
 6: end for
 7: if $n < N_D$ then
 8:     $pr \leftarrow false$
 9:     sendMotorPatternToRobot($\mathbf{p}^{(pred)}$)
10: else
11:     $pr \leftarrow true$
12: end if
13: return $pr$
```

[2] Sequence selection is done by the program logic.

5.4.7. Verification Logic

This module contains verification rules and provides information whether the current sensory-motor sample is valid (variable sv) and whether a goal point has been reached (variable gr). It also provides the sequence index i^*_{seq} and sample index i^*_{sam} for the next update cycle. The main purpose of the verification logic is to monitor the prediction error e_p that is a metric for the deviation of a physical action from an imagined action. If the prediction error is below a certain threshold, the architecture considers the observed sensory-motor sample as valid, otherwise as invalid. The observed samples that have been considered as valid are used as training data for both the MLP and the second-stage PAS.

The verification rules are written as an input-output mapping summarized in Table 22.

Module input: pr, e_p, e_g, $N_S(i_{seq})$, i_{seq}, i_{sam}	
Module output: sv, gr, i^*_{seq}, i^*_{sam}	
Parameters: thresholds e_p^{T}, e_g^{T}	
Input condition	**Output**
pr & $(e_p < e_p^{\mathsf{T}})$ & $(e_g \geq e_g^{\mathsf{T}})$ & $(i_{sam} < (N_S - 1))$	$sv \leftarrow 1, \quad gr \leftarrow 0, \quad i^*_{seq} \leftarrow i_{seq},$ $i^*_{sam} \leftarrow i_{sam} + 1$
pr & $(e_p < e_p^{\mathsf{T}})$ & $(e_g \geq e_g^{\mathsf{T}})$ & $(i_{sam} == (N_S - 1))$	$sv \leftarrow 1, \quad gr \leftarrow 0, \quad i^*_{seq} \leftarrow i_{seq} + 1,$ $i^*_{sam} \leftarrow 0$
pr & $(e_p < e_p^{\mathsf{T}})$ & $(e_g < e_g^{\mathsf{T}})$	$sv \leftarrow 1, \quad gr \leftarrow 1, \quad i^*_{seq} \leftarrow i_{seq} + 1,$ $i^*_{sam} \leftarrow 0$
pr & other	$sv \leftarrow 0, \quad gr \leftarrow 0, \quad i^*_{seq} \leftarrow i_{seq},$ $i^*_{sam} \leftarrow i_{sam}$
$\neg pr$ & $(e_g < e_g^{\mathsf{T}})$	$sv \leftarrow 1, \quad gr \leftarrow 1, \quad i^*_{seq} \leftarrow i_{seq} + 1,$ $i^*_{sam} \leftarrow 0$
$\neg pr$ & other	$sv \leftarrow 0, \quad gr \leftarrow 0, \quad i^*_{seq} \leftarrow i_{seq},$ $i^*_{sam} \leftarrow i_{sam}$

Table 22 Input-output mapping of the verification logic.

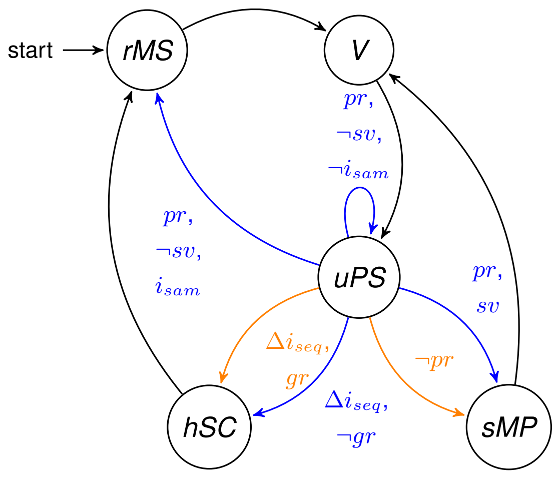
5.4.8. Program Logic

The program logic has the following purposes:

- Triggering a planning or replanning of sequences through a selected predictor.
- Controlling the execution of predicted sequences by sending the motor part to the robot.
- Extraction of valid visuo-proprioceptive samples during the interaction stage; these samples are used as training data for the generation of forward model and second-stage predictor.

- Monitoring of predicted sequences to warn in case of possible self-collision.
- Switching between predictors.

The program logic receives the currently observed visuo-proprioceptive sample with status data (i_{seq}, i_{sam}, pr), the output of the verification logic, and a given set of goal positions.
The program logic realizes a form of contextual control and is implemented by the following state machine:


start
rMS
V
uPS
hSC
sMP
pr, $\neg sv$, $\neg i_{sam}$
pr, $\neg sv$, i_{sam}
pr, sv
Δi_{seq}, gr
Δi_{seq}, $\neg gr$
$\neg pr$

In this state machine diagram, all state transitions from the state '*uPS*' are coloured blue or orange to point out that they depend on particular transition conditions. All remaining transitions are coloured black, meaning that those are fixed transitions independent of any condition. The coloured transition conditions represent the contextual state, in which the architecture is at any moment during operation. Each comma in the transition conditions represents a logical 'and'. The prerequisite for the starting state '*rMS*' is that the predictors PAS$_H$ and PAS$_A$ are trained (based on a previous constrained DOF exploration) and loaded, and that the head and arm are in home position, with the end-effector visible in the robot's FOV. Every state implements particular computations that I explain as follows.

5.4.8.1 State *rMS* ('*runMentalSimulation*')

In this state, a selected PAS is activated with an initial pattern $\mathbf{x}_{init} = [\mathbf{v}_f\ \mathbf{p}]^{(obs)}$ and a goal $\mathbf{v}_g$. The PAS runs with parameter κ and it predicts a visuo-motor sequence S bringing $\mathbf{v}_f$ closer to $\mathbf{v}_g$ through the corresponding motor part, see [221], [34] for details on the algorithm. The sequence S is written into the short-term sequence memory.
Then, the self-collision check is executed: The motor part of every predicted sample is extracted and the normalized DOF values are checked whether they are below a minimum or above a maximum value. For example, if the DOF of the elbow joint is close to a value representing a flexion in the elbow joint, then there is a risk of self-collision, since the hand may hit the trunk or the head. The motor part of the predicted sequences depend on the prediction

length factor κ determining the difference between subsequent DOF positions [221]. In other words, the parameter κ alters the resulting proprioceptive feedback.

5.4.8.2 State *V* ('*verify*')

In this state, the verification logic checks the current sample of the current sequence. The verification is implemented by the input-output mapping in Table 22.

5.4.8.3 State *uPS* ('*updateProgramState*')

In this state, architecture context state variables are updated depending on the results of the previous verification state. A sequence change is defined as $\Delta i_{seq} = i^*_{seq} - i_{seq}$ with $\Delta i_{seq} \neq 0$, and it represents a switching to another (action) trajectory. Querying a sequence change has precedence over querying the variables pr and sv. The following updates are done dependent on the contextual condition Γ:
If $\Gamma = \{\Delta i_{seq} \wedge \neg gr\}$:

$$n_{t/g} \leftarrow n_{t/g} + 1 \tag{58}$$

$$\xi \leftarrow \mathbf{x}_{init} \tag{59}$$

$$\Theta \leftarrow \text{'}\textit{hSC}\text{'} \tag{60}$$

The variable $n_{t/g}$ is the current number of trial per current goal. Vector ξ is the initial pattern sent to the PAS as input. Symbol Θ is the program state (see also Figure 46(b)).
If $\Gamma = \{\Delta i_{seq} \wedge gr\}$:

$$\{n_{t/g}, c_{ctr}, c\} \leftarrow \{n_{t/g} + 1, c_{ctr} + 1, c_{ctr}/N_{t/g}\} \tag{61}$$

$$\xi \leftarrow \mathbf{x}_{init} \tag{62}$$

$$\Theta \leftarrow \text{'}\textit{hSC}\text{'} \tag{63}$$

Note that in case of multiple updates denoted in one line, e.g. update (61), the order of updates proceeds sequentially from left to right, always taking the latest variable state into account. The variable c_{ctr} is the confidence counter and $N_{t/g}$ is the number of trials per goal. In addition, if sv holds true then the currently observed sample $[\mathbf{v}_f\ \mathbf{p}]^{(obs)}$ is recorded into the training data file for MLP and PAS$^{(2)}_{A}$ (see also Figure 45). In this case, it is the last sample of the sequence, thus a sequence end mark is also recorded.
If $\Gamma = \{pr \wedge sv\}$:

$$\mathbf{x}_{init} \leftarrow [\mathbf{v}_f\ \mathbf{p}]^{(obs)} \tag{64}$$

$$\Theta \leftarrow \text{'}\textit{sMP}\text{'} \tag{65}$$

In addition, the currently observed sample $[\mathbf{v}_f\ \mathbf{p}]^{(obs)}$ is recorded into the training data file for MLP and PAS$^{(2)}_{A}$.

If $\Gamma = \{pr \land \neg sv \land \neg i_{sam}\}$:

$$\{i^*_{seq}, i_{sam}, i^*_{sam}, pr, gr\} \leftarrow \{i_{seq}+1, 0, 0, \text{false}, \text{false}\} \tag{66}$$
$$\Theta \leftarrow \text{‘}\textit{uPS}\text{’} \tag{67}$$

This update terminates the execution of the current sequence, since already the first predicted sample ($i_{sam} = 0$) is not valid. The update prepares the switching to the next predicted sequence in the short-term sequence memory.
If $\Gamma = \{pr \land \neg sv \land i_{sam}\}$:

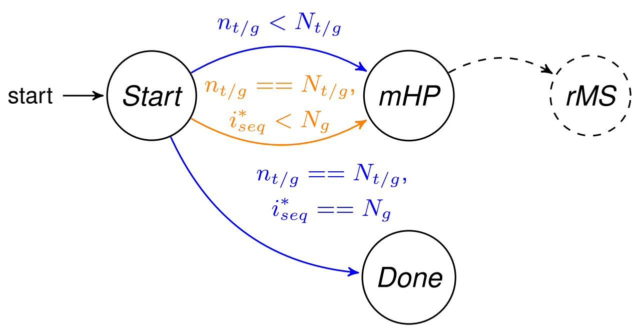
$$\{i^*_{sam}, i_{seq}, i_{sam}, pr\} \leftarrow \{0, i^*_{seq}, i^*_{sam}, \text{false}\} \tag{68}$$
$$\xi \leftarrow \mathbf{x}_{init} \tag{69}$$
$$\Theta \leftarrow \text{‘}\textit{rMS}\text{’} \tag{70}$$

If $\Gamma = \{\neg pr\}$:

$$\Theta \leftarrow \text{‘}\textit{sMP}\text{’} \tag{71}$$

5.4.8.4 State *hSC* (‘*handleSequenceChange*’)

In this state, a sequence change Δi_{seq} is managed by the following sub-state machine:



$n_{t/g} < N_{t/g}$
start
Start
$n_{t/g} == N_{t/g}$, $i^*_{seq} < N_g$
mHP
rMS
$n_{t/g} == N_{t/g}$, $i^*_{seq} == N_g$
Done

In the state ‘*mHP*’ (‘*moveLimbToHomePos*’), a particular limb is moved to a given home position. Here, the limb is the right arm. Before the limb is commanded into home position, the system variables are updated depending on the transition condition. Once the limb is in home position, there is a fixed transition into the state ‘*rMS*’ (‘*runMentalSimulation*’). In the state ‘*Done*’, the goals are recorded along with the final goal errors and confidence values, representing the acquired reachability map. This is a final state where the system halts.
Thus, from the state ‘*hSC*’, the system can either transit to the state ‘*rMS*’ (if not all goals and trials per goal have been explored yet) or come to a halt (‘*Done*’, if all goals and trials per goal have been explored). The following updates are done dependent on the contextual condition:

If $\Gamma = \{n_{t/g} < N_{t/g}\}$:

$$\{i^*_{seq}, i^*_{sam}, i_{seq}, i_{sam}, pr\} \leftarrow \{i^*_{seq} - 1, 0, i^*_{seq}, i^*_{sam}, \text{false}\} \tag{72}$$

$$\Theta \leftarrow \text{`}\textit{mHP}\text{'} \tag{73}$$

If $\Gamma = \{(n_{t/g} == N_{t/g}) \;\wedge\; (i^*_{seq} < N_g)\}$:

$$\{i^*_{sam}, i_{seq}, i_{sam}, pr, n_{t/g}, c_{ctr}\} \leftarrow \{0, i^*_{seq}, i^*_{sam}, \text{false}, 0, 0\} \tag{74}$$

$$\Theta \leftarrow \text{`}\textit{mHP}\text{'} \tag{75}$$

If $\Gamma = \{(n_{t/g} == N_{t/g}) \;\wedge\; (i^*_{seq} == N_g)\}$:

$$\Theta \leftarrow \text{`}\textit{Done}\text{'} \tag{76}$$

In case of $\Gamma = \{(n_{t/g} == N_{t/g}) \;\wedge\; (i^*_{seq} == N_g)\}$, the acquired reachability map is recorded into a file before the terminal state transition. The reachability map consists of a set of confidence values, each confidence c associated to a particular goal $\mathbf{v}_g$.

5.4.8.5 State *sMP* ('*sendMotorPatternAndObserveOutcome*')

In this state, the predicted motor pattern is sent to the robot. Two different transition conditions exist for entering this state: either $\Gamma = \{pr \wedge sv\}$ or $\Gamma = \{\neg pr\}$.

The selection and switching between the different PAS is still missing and will be implemented as future work. Up to now, the selection of the active PAS module is done manually.

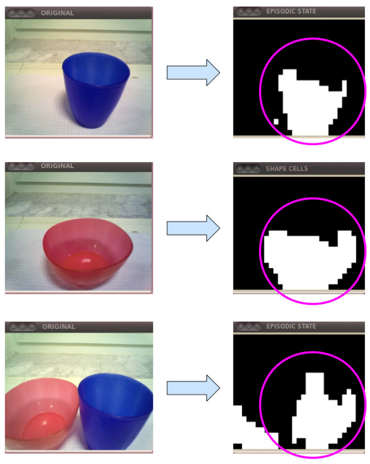
5.4.9. Motivator

The motivator module provides a set of goal positions to the program logic, which then delivers the selected goal to a selected PAS module. The purpose of the goal positions depends on the operational context: In general, they direct a PAS module; in particular, they direct the reaching phase and thus they influence the acquisition of the reachability map by clustering the workspace. Although the current implementation of this module only contains a pre-given list of goals, this module can be seen as (or replaced by) a sort of attention mechanism providing goals the robot should focus on.

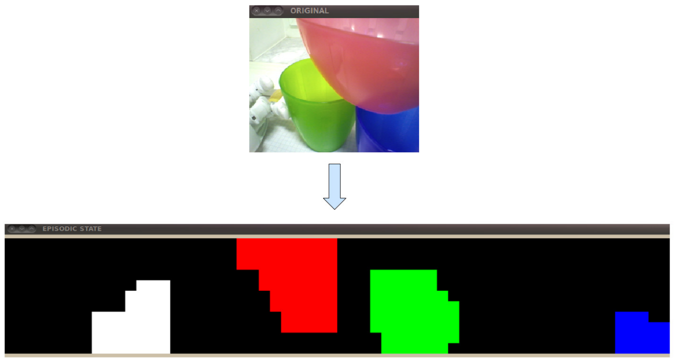
5.4.10. Useful Extension: Episodic Memory

Although not yet linked to the generation of actions, a useful extension is an episodic memory [219] that is encoded by a HHOP.
For the episodic memory, I adopted the HHOP [133], [39] that extends the original Hopfield network [73] by second-order (i.e. higher-order) connections in order to increase its storage efficiency. The improved storage efficiency is described in [133].
I implemented the HHOP according to the mathematical description in [133]. Compared to [133], [39], I extended the HHOP by increasing the visual features. Besides the shape or

contour of objects, their colour in RGB space is captured as well. The activations of the visual feature cells (Section 4.3.1) can be mapped directly to the HHOP neuron activation[3].
The benefit of including colour features is that different environmental situations can be better learned and recalled by the network. Figure 47 shows a comparative example of the HHOP with and without colour features.



(a) HHOP activation (right) *without* colour features.
HHOP size: 29×21 $(= 609$ neurons).



(b) HHOP activation (bottom) *with* colour features.
HHOP size: $15 \times 11 \times 4$ $(= 660$ neurons). Each feature field (shape, red, green, blue) has a size of 15×11.

Figure 47 Comparison of HHOP neurons with and without colour features. Figure 47(a) shows the current visual pattern to be learned by the HHOP *without* colour features, as used in [133], [39]. Different environmental situations cannot be distinguished, since objects with similar shape but different colour cause correlated memory patterns (indicated by the magenta circles). Figure 47(b) shows my proposed extension to the HHOP, with neurons sensitive to shape *and* also to *colour* features. Correlated memory patterns are significantly reduced, thus environmental situations can be better distinguished.

5.5. Experiments

5.5.1. Setup

In order to validate the proposed architecture, I conducted experiments on a NAO humanoid robot that was in a sitting posture. A marker was attached on its right arm for the visual detection of the arm tip. In the given home position, the head was oriented to the right arm such that the marker appeared roughly in the middle of the robot's FOV. Stereo vision is not possible on the NAO. For distance estimation, I used the extracted size of the shape of the colour marker when projected on the image plane of the top camera.

5.5.2. Ground Truth of Arm Tip Marker

The architecture does not contain any given kinematic model. In the architecture, positions are represented as normalized signals without physical units. In order to obtain a ground truth estimate of the arm tip marker and goal positions for experimental evaluation, the positions

[3] For this purpose, the image resolution of the visual feature cells needs to be reduced, otherwise too much computer memory (RAM) is used due to N^3 weights given N HHOP neurons.

(of goals and of arm tip) were transformed from the robot's FOV into the camera frame by using trigonometry. Then the points were transformed from the camera frame into the robot's torso frame by using the given kinematics of the NAO robot. Note that the NAO kinematics is only used to obtain a ground truth estimate in terms of positions relative to the robot's torso frame, it is *not* part of the architecture.

5.5.3. Selected Stages of Development

I validated selected stages of development, as presented in Figure 44.
Regarding stage 2, I have shown the results of the acquisition of early hand-eye coordination in Chapter 4. In particular, object tracking and evading by head control have been validated in Section 4.9. The acquired early reaching skill has been validated in Section 4.10.
Following the timeline in Figure 44, I focus on showing the skills of the stages 3 to 5.
An example of the conducted experiments on the NAO robot is summarized by the following video: `https://youtu.be/T3SgNKPZ3z4` that is a media attachment to the paper [222]. The video shows the stages 3 to 5, with the proprioceptive feedback already adapted to the robot. At the end of the video, the value of parameter κ altering this feedback was changed in order to demonstrate the prediction of self-collision in case of too high values of κ that lead to an increase of Δp.

5.6. Results

5.6.1. Constrained DOF Exploration

I specified the percentage of position deviation from the DOF home position. This position deviation was 5 % of the total range of the DOF with the greatest total range (among all DOF of a particular limb). During exploration, each DOF of a limb moves from its current angular position h into a target position $h \pm l$, with l computed by the aforementioned position deviation. Within this range, the difference between two consecutive angles sent was at least 2 % of the total range of a DOF. Sensory-motor samples are recorded in each of these steps while moving from the home position to either the upper or lower limit.
The robot started observing its arm tip marker when the arm was in home position (idle state). It then observed the arm tip position in its FOV while exploring the two degrees of freedom of the shoulder joint, followed by exploring the two degrees of freedom of the elbow joint, and finally by exploring the one degree of freedom wrist joint. Note that only one degree of freedom is moving at a time, the positions of all other degrees of freedom stay constant.
The constrained DOF exploration of the NAO arm has been shown in the previous chapter, with the complete results of all arm DOF visualized in Section 4.4.3.

5.6.2. Loops of Imaginary Trial and Physical Trial to Determine Optimal Proprioceptive Feedback, and to Avoid Self-Collision

With the training data obtained from the constrained DOF exploration stage in Section 5.6.1, the architecture is able to generate an imaginary sensory-motor sequence encoding reaching

for any (imaginary) goal point in the robot's FOV. Every imaginary sequence is then tried on the robot. The purpose of this stage is to find an optimal value range for the prediction length factor κ. For each DOF, κ determines the proprioceptive feedback Δp, causing changes in joint input.
The outcome of this imagination is the predicted number of reachable goals and the risk of self-collision due to hitting joint limits in the elbow joint (e.g. flexion of the elbow joint could cause collision of the lower arm with trunk or head). It is important to note that the system has no representation of the robot's morphology and its ability of accomplishing a higher resolution feedback, i.e. moving into joint positions where the position differences are small. Industrial robot arms can manage small position differences; on a UR5 arm as part of TOMM [52] for instance, $\kappa = 1$ would be enough to move [34]. Other robot platforms such as NAO [175] have difficulties resolving small position differences, for instance due to backlash in the joints.
In this stage, the motivator module yielded 36 imaginary goal points in the robot's FOV. Starting from the same home position for the end-effector, the architecture predicted a sensory-motor sequence for each imaginary goal point (encoding the reaching) with a given value of κ. This is referred to as *imaginary trial* or mental simulation. Every predicted sequence has then been executed on the robot physically, referred to as *physical trial*. The aim is to minimize the discrepancy between the number of goals reached in the imaginary trial and in the physical trial.
The physical trial revealed the number of goals actually reached with the given value of κ.
Moreover, the architecture accurately predicted self-collisions. The first physical self-collision occurred with $\kappa = 35$, which concurred with the prediction that self-collision will happen. To not damage the robot, higher values of κ have not been considered in the physical trials any more. The prediction results of the mental simulations concurred with the physical trials in so far that an optimal value range for κ was found to maximize reachability and to avoid self-collisions.
Figure 48 shows the predicted number of goals versus the physical number of goals reached depending on the value of parameter κ. A goal point was classified as reachable if the goal error e_g is less than 0.25. This corresponds to a ground truth value of less then 2.5 cm on the NAO robot.
Note that in a physical trial, one can expect that the number of reached goals is smaller than imagined in mental simulation. The reason for this is the sensory noise in the visual feedback and the robot's physical ability of resolving high resolution feedback (for example, $\kappa = 1...5$ will not cause motion in the joints of a NAO robot).
Corresponding to Figure 48, the predicted average goal error is shown in Figure 49.
Note that until now, κ is set manually. Automatic iteration over κ is future work, with the proposed method staying the same.
In sum, adaptation to a particular robot was accomplished by minimizing the discrepancy between the number of goals reached imaginary and physically while at the same time self-collision was avoided.

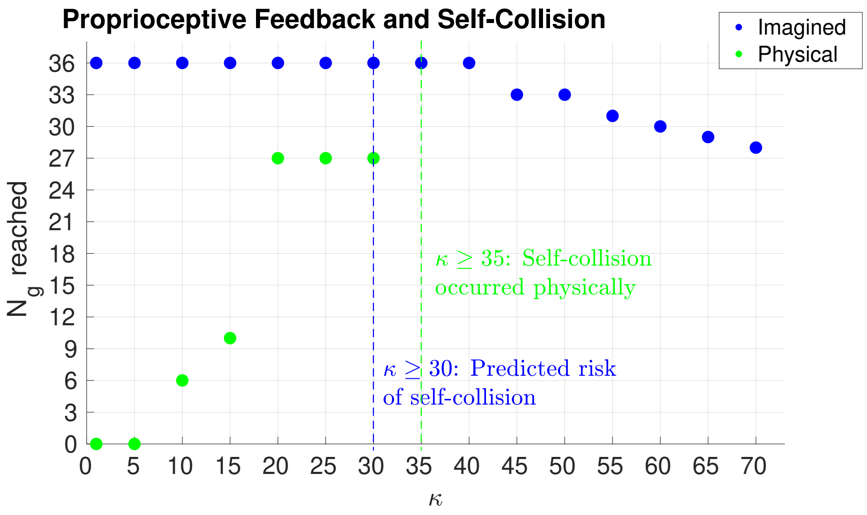


Figure 48 Mental simulation followed by physical trial to determine κ altering the proprioceptive feedback Δp causing changes in joint input, in order to maximize the overall reachability and to avoid self-collision. In the imagined trials, the system predicted self-collision of the robot arm with head or trunk when $\kappa \geq 30$. In the physical trials, self-collision occurred with $\kappa \geq 35$; avoiding self-collision has higher priority than the number of goals reached, thus the number of goals reached is neglected for values of κ causing self-collision.

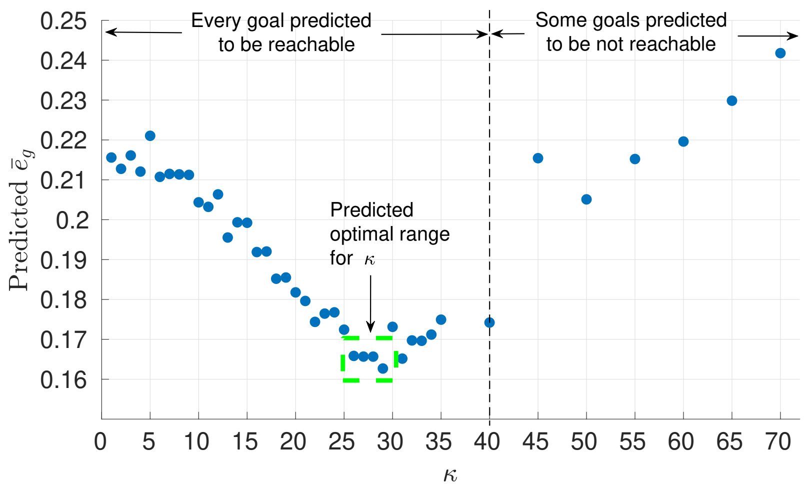
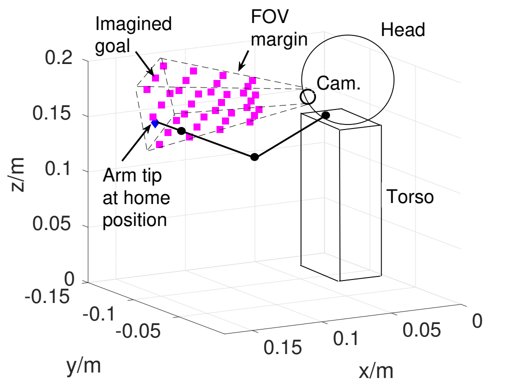


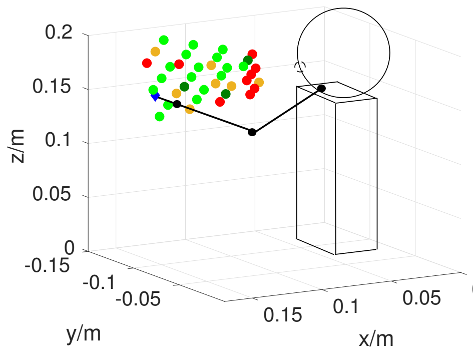
Figure 49 Predicted average goal error $\bar{e}_g$ over κ after every imaginary goal was reached in the mental simulation (i.e. imaginary trial), additional result to Figure 48. The dashed green box indicates the predicted optimal range for κ, since in that range the predicted average goal error is minimal. The physical trials (Figure 48) have confirmed this range.

5.6.3. Acquisition of an Initial Reachability Map

By using the PAS for arm control, the robot now tries to reach each goal position. Every reaching attempt starts from the same home position shown in Figure 50(a). The outcome is the acquisition of an initial reachability map, see Figure 50(b). This reachability map represents how well the robot can cover its (local) workspace by its current predictor. A goal point was classified as reachable if the goal error e_g is less than 0.25. In the NAO robot's torso frame, this value corresponds to a ground truth of less then 2.5 cm. Every imaginary goal point (Figure 50(a)) was attempted to be reached three times from the same home position and the outcome was represented by a confidence score (visualised by the colours in Figure 50(b)).



(a) Imaginary goals (36) in the FOV of the robot using a monocular camera (e.g. top camera of the NAO).



(b) Initial reachability map: 3D view in torso frame, arm shown in home position

Figure 50 Acquisition of an initial reachability map 50(b) from the start setup in 50(a). Starting from home position, the robot tried to reach every goal multiple times (in this example, three times per goal). The system then self-evaluated its success by assigning a confidence score to each goal. Red colour indicates goal points that the robot could not reach. All other colours indicate successfully reached goals with different confidence levels; orange indicates low confidence (33 %, meaning that one of three trials was successful), dark green indicates medium confidence (67 %, two of three trials successful), and bright green indicates high confidence (100 %, three of three trials successful).

The mean goal error $\bar{e}_g$ describes how well one particular goal point can be reached if multiple trials are conducted. The number of trials per goal is denoted by $N_{t/g}$ and was set to 3. The mean goal error is defined as $\bar{e}_g = (\sum_{i=1}^{N_{t/g}} e_g^{(i)})/N_{t/g}$ with i as trial index.

For the goals that could be reached with 100 % confidence (i.e. 3 of 3 reaching trials successful), the average value of $\bar{e}_g$ was 0.1943. This corresponded to a measured value of 1.82 cm in the torso frame of the NAO robot. On the NAO, a deviation of 1.82 cm between goal point and arm tip marker represents a successful reach.

5.6.4. Generation of a Forward Model

During the acquisition of the initial reachability map shown previously, the architecture verified the prediction error and obtained training data for the generation of a forward model. The training data consisted of 299 self-verified samples. In order to investigate the effect of the hidden layer size influencing the network generalization ability, four different MLP configurations were used. Each of them had the same number of input neurons $N_{in} = 5$ encoding the proprioceptive vector of the arm, and the same number of output neurons $N_{out} = 3$ encoding the predicted position of the (imaginary) arm tip in the FOV. The difference between the MLP configurations is shown in Table 23. The configurations 1 and 2 represent a tradeoff between

Parameter and metric	**Configuration 1**	**Configuration 2**	**Configuration 3**	**Configuration 4**
N_h	12	7	17	5
MSE	$5.2 \cdot 10^{-4}$	$6.2 \cdot 10^{-4}$	$5.141 \cdot 10^{-4}$	$7.002 \cdot 10^{-4}$

Table 23 Difference between the MLP configurations.

MSE, generalization, and learning time. Configuration 3 was chosen to investigate the effect of more than three times the number of input neurons, which would lead to a severe overfitting and relative long learning time (more than 6 hours on an i7 CPU). Configuration 4 was chosen to investigate the minimum number of neurons the hidden layer should have relative to its input layer ($N_h = N_i$). For the configurations 3 and 4, the learning was stopped if the difference in the MSE after 10000 epochs was smaller than $1.0 \cdot 10^{-8}$. According to a rule of thumb for choosing the number of hidden neurons [134], [85], configuration 2 would yield the ideal performance.

5.6.5. Evaluation Metric for the Acquired Predictor Models

The purpose of this evaluation metric is to provide a means for the system to self-evaluate the performance of its acquired predictors: either a single PAS (first-stage or second-stage, see the predictors with the labels 1, 2, 4 in Figure 45) or a PAS supported by a forward model (see the series connection by predictors 3 and 2 in Figure 45).
I investigated the prediction accuracy when the robot is acquiring the initial reachability map under simulated all time covering of the arm tip. For this scenario, the MLP configurations 1 and 2 were chosen, since they represent the tradeoff configurations. The following principle stays for any other configuration. I used the following metrics:

- Predicted goal error: $e_g^{(p)} = \|\mathbf{v}_g - \mathbf{v}_{AT}^{(p)}\|$.
- Measured goal error: $e_g^{(m)} = \|\mathbf{v}_g - \mathbf{v}_{AT}^{(m)}\|$.

The deviation d between the mean predicted goal error and the measured goal error was investigated to evaluate the prediction accuracy. The average absolute deviation $|\bar{d}|$ across all reachable goals $N_g^{(r)}$ is defined as $|\bar{d}| = \sum_{j=1}^{N_g^{(r)}} |d_j|/N_g^{(r)}$. The average absolute deviation $|\bar{d}|$ was 7.53 % for MLP configuration 1 and 8.41 % for MLP configuration 2. The number of reached goals $N_g^{(r)}$ was 10 for MLP configuration 1 and 12 for MLP configuration 2.
An additional metric for evaluation of the overall reaching performance is the *cumulative confidence* that I define as $C = \sum_{j=1}^{N_g} c_j \text{ with } c_j = z_j/N_{t/g}$. Here, for a particular goal j, the variable z_j is simply a counter that will be increased if j is reached, z_j corresponds to the variable c_{ctr} in update rule (61) in Section 5.4.8. The parameter $N_{t/g}$ is the given number of trials per goal. The cumulative confidence C is obtained by adding up the confidence values

c_j over all goals and it represents the reaching outcome per given configuration. I propose this metric for self-verification whether one particular predictor configuration (either a PAS or a PAS supported by a MLP) is better than another one.

Table 24 provides a comparison between the support by MLP configuration 1 and 2 in terms of accuracy and the number of goals reached (along with the confidence). In this example,

	MLP configuration 1		MLP configuration 2	
Goal No.	Deviation of $\bar{e}_g^{(p)}$ from $\bar{e}_g^{(m)}$	c	Deviation of $\bar{e}_g^{(p)}$ from $\bar{e}_g^{(m)}$	c
1	-14.09 %	0.33	$+0.40$ %	0.33
2	-0.15 %	0.67	$+28.65$ %	0.33
15	-6.33 %	0.33	-0.94 %	0.33
17	0.0 %	0.33	–	–
22	–	–	$+19.07$ %	0.67
23	–	–	-5.06 %	0.67
26	–	–	-11.09 %	0.33
27	$+3.07$ %	1.0	-8.84 %	0.33
30	–	–	-10.34 %	0.67
31	-10.57 %	1.0	-1.63 %	1.0
32	-19.27 %	1.0	$+7.63$ %	1.0
33	$+12.45$ %	0.33	–	–
34	-4.96 %	1.0	-2.82 %	1.0
35	-4.38 %	1.0	-4.42 %	1.0
	MLP configuration 1		MLP configuration 2	
Cumulative c	6.99		7.66	

Table 24 Deviation of the predicted goal error from the measured goal error, simulated all time covering of arm tip, i.e. simulated visual occlusion during the entire interaction.

there was only marginal difference predicted between the performance of MLP configuration 1 and 2. In general, the higher the cumulative confidence is, the better the performed skill, in this case the reaching.

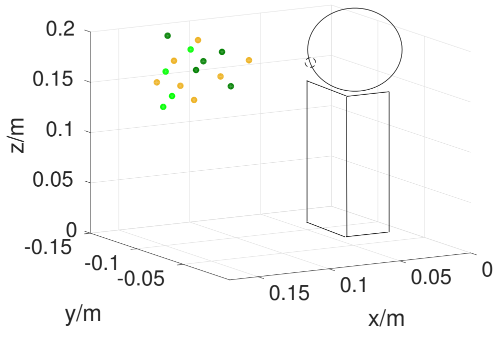
5.6.6. Development of Reachability Map under Disturbance

For this purpose, I investigated different covering lengths of the robot's arm tip. In this context, "covering length" means the temporal length, i.e. the duration for which the arm tip is covered and thus *not* visible for the robot's camera. Another expression for covering length is

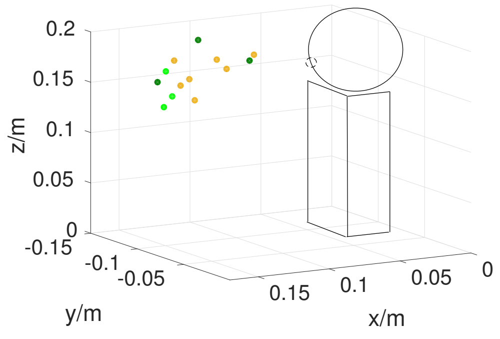
"occlusion length", referring to a full visual occlusion with different durations. For each covering length, I validated the acquisition of the reachability map with and without the support by different forward model configurations.
I used my proposed cumulative confidence as a metric for evaluation and validated the acquisition of the initial reachability map under environmental disturbance that is always a complete covering of the robot's arm tip but for different durations. Five different cases were validated: acquisition of reachability map without any forward model support, and acquisition of reachability map with support by *MLP* configuration 1, configuration 2, configuration 3, and configuration 4, respectively. For each of these cases, I investigated different covering lengths: 67 %, 75 %, 86 %, 100 % of the total interaction time. For example, a covering length of 75 % means that only every 4th incoming visual sample contained an observable arm tip position, a covering length of 86 % means that only every 7th incoming visual sample contained an observable arm tip position.
Due to these occlusion lengths and an optional support by a forward model, the robot acquired different reachability maps that contain the reached goals for three selected cases:

1. no forward model support (see Figure 51),
2. support by MLP configuration 1 (see Figure 52), and
3. support by MLP configuration 2 (see Figure 53).

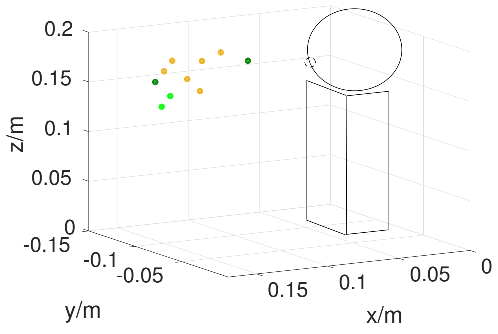
Figures 51, 52, 53 show that the total number of goals reached with the support by the acquired forward model (either MLP 1 or MLP 2) was always bigger than without the support (except for 67 % covering where the support by MLP 2 yielded the same number of reached goals as without support). The longer the covering length, the better the support by a forward model pays off. The reachability map resulting from the support by MLP configurations 3 and 4 was not visualized, since the results were marginally different from configurations 1 and 2.
The results for all the different cases are summarized by Figure 54.



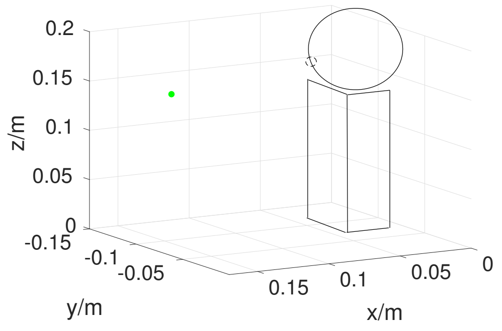
(a) Covering 67 % of total interaction time: no forward model support; goals reached: 16



(b) Covering 75 % of total interaction time: no forward model support; goals reached: 13

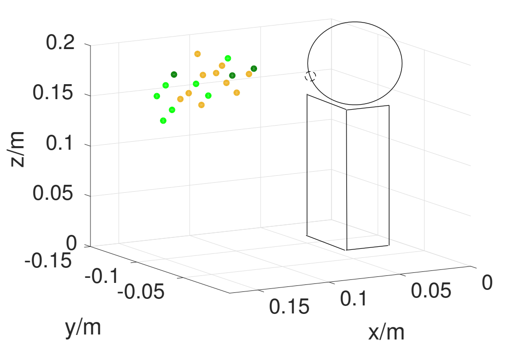


(c) Covering 86 % of total interaction time: no forward model support; goals reached: 10

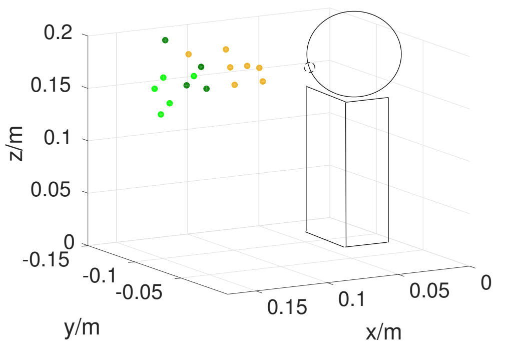


(d) Covering 100 % of total interaction time: no forward model support; goals reached: 1

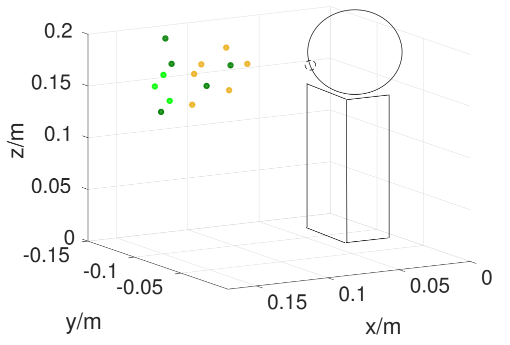
Figure 51 Acquired reachability map under disturbance, PAS$_A$ without any support by forward model. Without such support, the robot can reach almost none of the goals if the arm is occluded or covered during the *entire* interaction (Figure 51(d)).



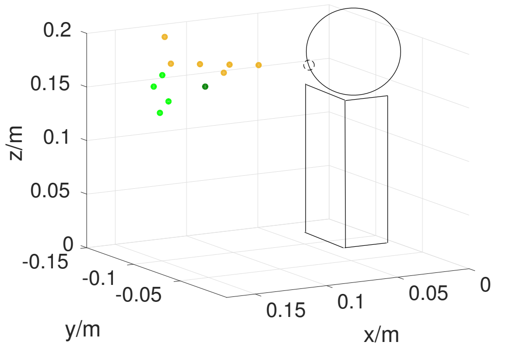
(a) Covering 67 % of total interaction time: supported by MLP 1; goals reached: 20



(b) Covering 75 % of total interaction time: supported by MLP 1; goals reached: 16

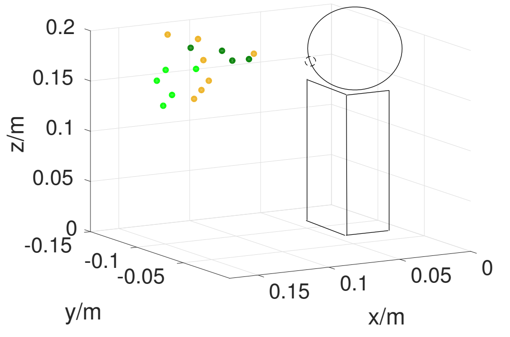


(c) Covering 86 % of total interaction time: supported by MLP 1; goals reached: 14

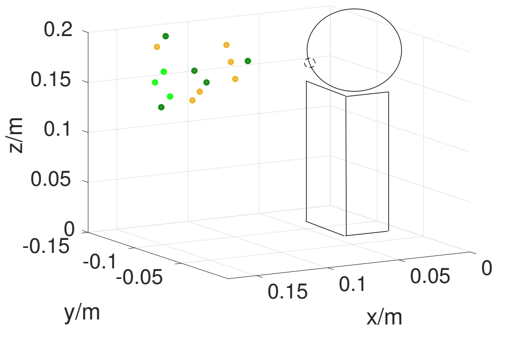


(d) Covering 100 % of total interaction time: supported by MLP 1; goals reached: 11

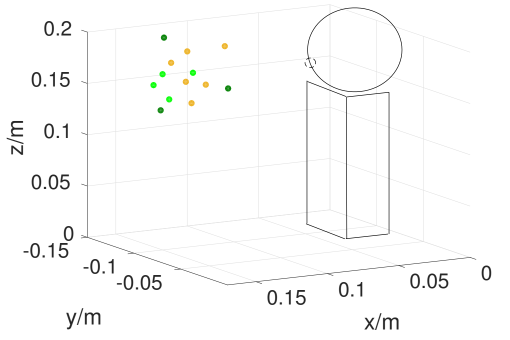
Figure 52 Acquired reachability map under disturbance, PAS$_A$ was supported by MLP 1. Even if the arm is occluded or covered during the *entire* interaction (Figure 52(d)), the robot can still reach approximately one third of the goals (11 of 36).



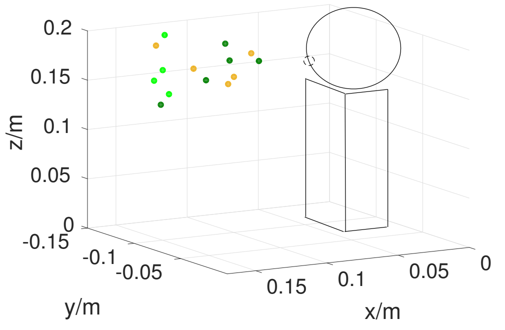
(a) Covering 67 % of total interaction time: supported by MLP 2; goals reached: 16



(b) Covering 75 % of total interaction time: supported by MLP 2; goals reached: 14



(c) Covering 86 % of total interaction time: supported by MLP 2; goals reached: 13



(d) Covering 100 % of total interaction time: supported by MLP 2; goals reached: 14

Figure 53 Acquired reachability map under disturbance, PAS$_A$ was supported by MLP 2. Even if the arm is occluded or covered during the *entire* interaction (Figure 53(d)), the robot can still reach more than one third and almost half of the goals (14 of 36).

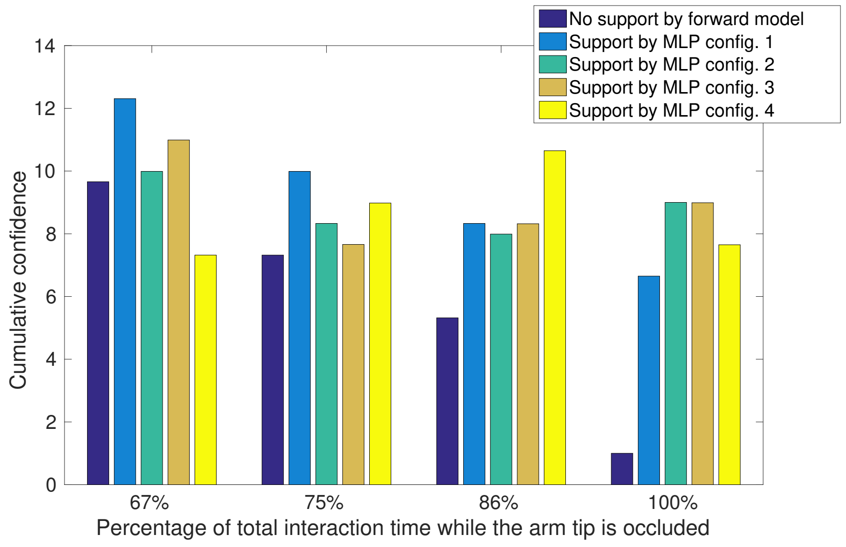


Figure 54 Cumulative confidence depending on forward model support. There is only a marginal difference between the forward model configurations, independent of the occlusion or covering length. Nevertheless, one can see a difference in the acquisition of the reachability map with and without support by any forward model in cases of long term coverings of the robot's arm tip (75 % or more of total interaction time). In such cases, the performance significantly deteriorates without a forward model support.

5.6.6.1 Detailed Quantitative Results per Case

The detailed quantitative results for each case are provided by the Tables 25, 26, 27, and 28. In these tables, “No MLP” means no support by any forward model at all, whereas “MLP x” means support by MLP configuration x, with x either 1, 2, 3, or 4.

c	N_g reached depending on MLP support:				
	No MLP	MLP 1	MLP 2	MLP 3	MLP 4
1.0	4	7	5	3	4
0.67	5	3	4	7	2
0.33	7	10	7	10	6
Cumulative c	9.66	12.31	9.99	10.99	7.32

Table 25 Physical reaching outcome: arm tip covered during 67 % of the total interaction time.

c	N_g reached depending on MLP support:				
	No MLP	MLP 1	MLP 2	MLP 3	MLP 4
1.0	3	5	3	4	3
0.67	3	4	5	3	4
0.33	7	7	6	5	10
Cumulative c	7.32	9.99	8.33	7.66	8.98

Table 26 Physical reaching outcome: arm tip covered during 75 % of the total interaction time.

c	N_g reached depending on MLP support:				
	No MLP	MLP 1	MLP 2	MLP 3	MLP 4
1.0	2	3	4	6	8
0.67	2	5	3	1	1
0.33	6	6	6	5	6
Cumulative c	5.32	8.33	7.99	8.32	10.65

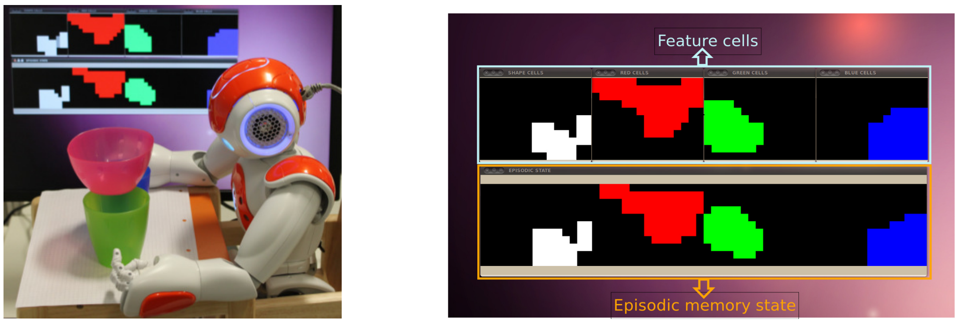
Table 27 Physical reaching outcome: arm tip covered during 86 % of the total interaction time.

c	N_g reached depending on MLP support:				
	No MLP	MLP 1	MLP 2	MLP 3	MLP 4
1.0	1	4	4	5	4
0.67	–	1	5	3	2
0.33	–	6	5	6	7
Cumulative c	1	6.65	9	8.99	7.65

Table 28 Physical reaching outcome: All time covering of arm tip, i.e. during the entire interaction.

5.6.7. Episodic Memory — Distinction between Environmental Situations

In order to evaluate the proposed HHOP encoding the episodic memory (Section 5.4.10), I validated the learning and recall of a set of different environmental situations. The set consisted of 7 different environmental situations, one of them shown in Figure 55. Each



(a) Environmental situation perceived by the visual feature cells and learned and recalled by the episodic memory

(b) Recalled episodic memory state (bottom window), given the activation of visual feature cells (top window)

Figure 55 Environmental situation learned by the proposed HHOP encoding episodic memory. Figure 55(a) shows the NAO sitting in a chair and facing three cups. Corresponding to this situation, the graphical user interface in Figure 55(b) shows the current state of the visual feature cells fed into the episodic memory and the resulting recalled memory state.

environmental situation is grounded in a sensory pattern that is stored into the HHOP.

The environmental situations were: 1) Red cup, 2) green cup, 3) blue cup, 4) red and green cup, 5) red and blue cup, 6) green and blue cup, and 7) red and green and blue cup.

The learning and recall of this set of situations was done for two cases: colour features enabled and colour features disabled.

I used the correlation factor as quantitative criterion in order to evaluate the discrimination of patterns. If the correlation factor between the recalled pattern and the stored pattern is greater than 80 %, then the recall is correct.

With the *colour feature disabled*, there is no chance to separate objects having similar shape but different colour. The shape feature as the only type of feature resulted into recalled patterns that were highly ambiguous.

With the *colour feature enabled*, all the aforementioned 7 situations could be separated with

a correlation factor greater than 90 %. More situations were evaluated with enabled colour features. Pattern interference or ambiguity started when more than 15 different situations were stored into the HHOP, causing a decrease in the correlation factor below 80 %.
An example of the conducted experiments with the HHOP on the NAO robot is summarized by the following video: `https://youtu.be/XoJHJKzSx2w` that is a media attachment to the paper [219]. The video shows the learning and recall of different environmental situations by the HHOP.

5.7. Discussion

A traditional forward model would map both the current sensory input and the current motor input to a sensory output at the next time step. In the proposed architecture, the forward model was kept simple: The MLP only mapped a motor input to a visual (i.e. sensory) output. This is sufficient, since this forward model was meant to only support the spatiotemporal predictor, and not to replace it. When applied in the series circuit (i.e. MLP output connected to PAS visual input as shown in Figure 45), the reaching performance reflected by the cumulative confidence showed only marginal differences when the number of hidden neurons was altered in the MLP.
A limitation of the proposed architecture is that the selection or switching between the different PAS is not fully implemented yet. Finalizing the predictor switching inside the program logic would result into the exploration of the entire workspace, and not only of the current space within the robot's FOV. This can be established by controlling the head by PAS$_H$ to make it follow the arm tip. The motor input of the MLP can be scaled up accordingly to include the DOF of the head as well in order to resolve ambiguities in the learned mapping. Completing this predictor switching is immediate future work. Nevertheless, the already established mechanisms implementing the loops of imaginary trial and physical trial would be re-used at any future stage. Thus, they provide the basis for scalability on the skill level.
I investigated each developmental stage separately from the others in order to analyse and validate the corresponding mechanisms. Based on the gained insights, an extension would be an algorithm that connects my proposed developmental stages together, along with my proposed verification metrics. This would lead to an automatized switching between developmental stages.
A limitation of the self-collision detection is that it is restricted to joints that are actively controlled.
In sum, I investigated a variety of capabilities and skills resulting from my architecture, with particular focus on skills that are autonomously acquired, i.e. in the absence of a human teacher.
An imitation skill is future work and out of scope of this thesis. An appearance level imitation [95] is a foreseeable skill that would potentially result from my architecture design. For example, if an external object is moved in front of the robot, the motivator module or a stimulus attractor would capture the trajectory of that object, representing a series of visual goal

positions. Each time the robot arm approaches a goal position, the home position would be updated to be the current position. The resulting behaviour would be a primitive, appearance level-based imitation.
The episodic memory encoded by the HHOP network can be used to trigger goal-directed actions depending on the environmental situation. The inclusion of colour features yielded a better differentiation of environmental situations than with the shape features alone.

5.8. Summary

In this chapter, I proposed a cognitive architecture that combines principles of self-verification with multi-stage, multi-purpose predictors for robust bootstrapping of skills. I showed how this architecture supports particular cognitive capabilities and investigated the robot (sensory-motor) skills that emerge from these capabilities.
Any robot skill, be it object tracking, evading, or reaching, is composed of visuo-motor sequences directed to given goal points in the visual space. These sequences are embedded into loops of imaginary trial and physical trial, with the prediction error as metric triggering a new loop. I demonstrated the benefit of this mechanism by validating the corresponding stages of development: self-collision prediction depending on the proprioceptive feedback, autonomous extraction of self-verified training data for subsequent generation of forward model, and the resulting improvement of the reaching skill.
One of the cognitive capabilities was the acquisition of an internal representation of predictor performance. This representation was formed by confidence scores that were assigned to the goals after self-verifying the success of the physical actions.
In my demonstrated example of the reaching skill, the confidence scores formed a reachability map and the cumulative confidence was used as a metric for the predictor performance supporting this skill. The forward model was implemented by a spatial predictor (i.e. MLP) and aimed at supporting my spatiotemporal predictor (i.e. PAS) through a series connection between them. The forward model was trained with 299 pattern samples autonomously obtained from a previous reachability map acquisition. Although the PAS itself has the ability of short-term prediction, I validated that the support by a forward model is beneficial for long-term occlusions of the robot's arm tip, e.g. if the arm tip is occluded for more than 75 % of the total interaction time. Therefore, the support by a forward model improves the prediction capability of the overall system.
An important feature of my proposed architecture is its capability to allow the robot to adapt to its body. This adaptation emerges from the cross-platform applicability of the PAS validated on the robots NAO and TOMM as well as from the adjustment of proprioceptive feedback through loops of imaginary and physical actions.

6. Chapter

Conclusion

In this thesis, I proposed the SVCA as a new cognitive architecture for autonomous sensory-motor skill acquisition. The main characteristics of the proposed architecture are spatiotemporal learning and prediction as well as adaptability and robustness to sensory-motor deficits. These characteristics were successfully validated on the robots NAO and TOMM.
The conclusion provides a final summary that reviews the proposed methods and the achievements. An outlook finishes this thesis by addressing current limits of the architecture and possible future work.

6.1. Final Summary

6.1.1. Overall Problem and Scope

The overall problem addressed in this thesis was the *autonomous acquisition of sensory-motor coordination skills*, where the term *skills* has been defined to comprise meaningful robot behaviour, internal representations, and cognitive capabilities that in turn facilitate and improve the behaviour. More precisely, the aim was to make a humanoid robot learn fundamental sensory-motor abilities similar to those of infants in their first weeks after birth. This work is settled in the domain of *developmental robotics*, since the focus was to investigate aspects of *learning using minimum prior knowledge or abilities*. In particular, this aim is related to the problem of *poverty of stimulus* [192, p. 260], [193], which constrains developmental biological and artificial agents regarding the amount and complexity of data they can sample from their local environment, when they are in the initial stages of their lifespan.
The scope of this thesis is in the *domain of sensory-motor coordination*. Learning was accomplished through methods of self-exploration and self-verification, excluding a human teacher. Hence, further higher-level skills, such as recognizing and opening a door, were out of scope of this thesis. Imitation learning was also out of scope.

6.1.2. Key Mechanism, Problems, and Contributed Solutions

Based on the problem of poverty of stimulus, I formulated a set of research questions that guided my work in this thesis.
My review on the literature yielded that the most promising state-of-the-art methodology for developmental agents is the *predictive coding framework*, since it encompasses three cognitive capabilities: 1) learning, 2) prediction, and 3) recognition of sensory-motor sequences. This has also been substantiated by recent neuroscientific evidence that prediction is a key operating mechanism of the neocortex.

For this reason, I adopted the MTRNN [230] to be the basic method used for learning within the proposed architecture.

I contributed to the original MTRNN design by proposing a *modified version of the MTRNN* that does not require additional neural networks (e.g. SOMs) for pre- and postprocessing the network input and output. This was accomplished by replacing the neural units working with softmax activation by units working with sigmoid activation, making the entire network consist of sigmoid units only. The sigmoid activation yielded uniformity, continuity of signals, and allowed *pre- and postprocessing* by using a proposed *analytical* schema instead of training further networks.

To avoid overfitting, a further contribution was the addition of an *early stopping* schema as part of the BPTT training method.

In addition to this modified MTRNN, I proposed an *AHE* in order to reduce the parameterization effort of the MTRNN. Successful learning by the MTRNN is dependent on the chosen hyperparameters, especially on the timescales of the different neural groups. For the AHE, I adopted the SA-DE [28]. This resulted in my proposed *EO-MTRNN* that estimates all neural timescales and also the number of neurons per context group in order to successfully learn a given set of teaching data. In single and multiple sequence training cases, the results showed that the EO-MTRNN yields an improvement of approximately 43 % in learning the data, compared to a network configuration with non-optimized hyperparameters. When multiple sequences are trained simultaneously, the network adjusts the connective weights common to all sequences and distinguishes between them by self-organizing different initial context states corresponding to each sequence.

If a predictive coding method is used within a developmental agent, one important problem remains, however. The literature review of existing work about predictive coding including the MTRNN resulted in the fact that the teaching data have been provided by an external intelligence, e.g. a human teaching the robot. But since my focus was on an *autonomous* acquisition of sensory-motor skills, I proposed a method that self-generates the teaching data, with consideration of the poverty of stimulus. This implied that the teaching data cannot be rich in terms of its amount and complexity.

Thus, I proposed to *model the poverty of stimulus* by conceiving a rather simple sensory-motor interface to a humanoid robot. The interface extracts position as the only physical quantity in the visual space and in the motor space. Any sensory-motor sample that is collected by the system contains the position of a feature of interest in the robot's FOV as well as motor positions.

Based on this interface, I proposed the method of *constrained DOF exploration* for an autonomous generation of the first set of teaching data. The constrained DOF exploration used a simple position control to move each DOF one after the other within a limited range that was only 10–20 % of the total range of a DOF. During each DOF motion, the method recorded the position of a visual feature in the robot's FOV along with the corresponding motor positions of the limb being explored. These sensory-motor samples were the first set of autonomously generated teaching data that were the basis for subsequent learning.

A further problem was that existing predictive coding frameworks barely addressed *coordina-*

tion skills, i.e. sensory-motor skills related to moving objects the robot is supposed to interact with. In such cases, a coordination skill has to take account of the motion of self-features (e.g. own body parts) as well as the motion of external features (e.g. objects). Existing predictive coding methods can indeed capture procedural skills well, such as pushing and lifting a stationary object for instance, coping with a variance of object position within a few centimeters. However, they were limited in their ability to learn to coordinate the robot interacting with external moving objects. In some of the related works, for example in [230], [126], the skill of tracking an object by head to support a lifting task was given by a pre-programmed vision tracking system.

I contributed to solve this problem by proposing algorithms that generalize from a few sensory-motor data samples[1] to meaningful coordination behaviour that is tracking, evading, and reaching to stationary as well as to moving targets. My proposed algorithms formed a system that I refer to as *PAS*. The PAS uses the proposed EO-MTRNN as its key method. The proposed PAS blends together two fundamental aspects in sensory-motor coordination: 1) learning and prediction of self-motion, and 2) learning and prediction of external object motion. The PAS integrates action generation and action selection into one framework. This allows a greater flexibility in movement generation compared to the state-of-the-art action selection like in the iCub cognitive architecture. The prediction capability of the PAS increases the robustness of coordination skills, for instance predictive motions despite environmental disturbances like occlusion or covering of the target object. Feature prediction ranged from 150–450 milliseconds ahead in time. In the experiments, the PAS learned to control the 2 DOF head of the NAO robot from 48 sensory-motor samples. The resultant skills were object tracking and evading, with a tracking accuracy of 1–2 % as remaining visual error. The PAS learned to control the 5 DOF arm of the NAO robot from 126 sensory-motor samples. The resultant skill was an early reaching ability, with the arm tip location predicted up to 150 milliseconds ahead in time at 20 Hz. Moreover, the PAS learned to control a 6 DOF arm of the TOMM robot[2]. The PAS was able of multi-staged reaching, where the first-stage bootstrapped training data for the second-stage. Given a set of evenly distributed goal positions in the robot's workspace, the second-stage reaching could reduce the average reaching time by 21 % for 33 % of the goals, i.e. for one third of the workspace.

While the PAS successfully yielded autonomous learning of sensory-motor coordination based on a minimum amount of self-generated teaching data, its prediction capability was restricted to short-term durations, i.e. approximately 1–2 seconds.

In order to extend the prediction capability for multiple purposes, I connected the PAS to a MLP and to rule-based algorithms. With the PAS as main building block, this resulted into a new *developmental cognitive architecture* — the *SVCA* — that uses the proposed *LITAPT*. The LITAPT verify sensory-motor data in order to 1) generate a forward model improving the sensory-motor skill in terms of robustness, 2) adapt to different robots by adjusting the proprioceptive feedback, and 3) avoid self-collision. The architecture improved in a coordination skill. Long-term prediction capability was acquired through the support of the PAS by

[1] that were collected by the aforementioned constrained DOF exploration

[2] The TOMM robot has two UR5 industrial arms.

a forward model MLP that was trained based on self-generated interaction experience. For instance, reaching was improved to blindfolded reaching to (stationary) targets, with visual occlusion that can last for minutes up to hours.

6.1.3. Capabilities of the Proposed Cognitive Architecture

The capabilities of the proposed SVCA can be summarized as follows:

1. Since the architecture is built on the PAS, its main capability is *prediction* along with the *generation of sensory-motor skills* that were object tracking and evading by head, and reaching by the arm. These skills are autonomously bootstrapped from a minimum amount of training data.

2. The LITAPT yield self-verification of sensory-motor samples that are used to *generate a forward model* supporting the PAS. This yields a skill execution (here it was reaching) that is *robust* even under a complete loss of visual data for the entire interaction time, no matter how long, i.e. no matter whether minutes or hours. In case of a complete loss of visual data (e.g. cameras failing or cameras covered), the reaching is restricted to stationary targets.

3. The LITAPT yield the *adjustment of the proprioceptive feedback* that is important to *adapt* to different robot platforms including imprecise robots with backlash in the joints or motors. The loops consisted of reaching actions to imaginary goal positions. In each loop, the architecture simulated the reaching and predicted the number of goals reached (imaginary trial). Then it verified the simulated action in real through physical execution on the robot (physical trial). Each loop was conducted for a particular parameter value influencing the proprioceptive feedback. The minimisation of the discrepancy between the number of goals reached imaginary and physically yielded an optimal range for the proprioceptive feedback, along with the avoidance of self-collision that was predicted as part of the imaginary trials.

6.1.4. Insights on the Autonomous Acquisition of Sensory-Motor Skills

In sum, my contributions led to the following insights:

1. The key mechanism is predictive coding. It can be re-used in a multi-purpose manner for learning, prediction, and recognition of sensory-motor patterns.

2. The proposed AHE reduces the efforts to parameterize a predictive coding method like the MTRNN. The AHE yields an improvement of approximately 43 % in the learning performance compared to a non-optimized network configuration.

3. Aspects of poverty of stimulus can be modelled through a sensory-motor interface extracting positions in the visual and motor space as the only physical quantity. The proposed constrained DOF exploration provides the first set of sensory-motor data

generated by the agent itself.

4. The proposed PAS approaches the problem of poverty of stimulus. The PAS extends a predictive coding method by generalizing to sensory-motor skills from a minimum amount of sensory-motor data generated by the developing agent itself through constrained DOF exploration.

5. The PAS can improve in a sensory-motor skill, such as reaching, by reducing the reaching time through multi-staged learning and prediction.

6. The PAS is a building block of the proposed SVCA that uses LITAPT in order to adapt to different robot platforms, to avoid self-collision, and to improve the sensory-motor skills. For instance, a robust execution of reaching can be accomplished under disturbed environmental conditions like long-term absence of visual data.

7. The PAS and the cognitive architecture built on it can operate on different robot platforms.

6.2. Outlook

This thesis showed how a predictive coding method can be optimized (by AHE) and extended to a new developmental cognitive architecture. The proposed methods for autonomous bootstrap learning of sensory-motor coordination with its subsequent improvement and the proposed method for adapting to morphological conditions, such as motor backlash, are all together conducive to future autonomous agents that are characterized by sensory-motor development, adaptability, and robustness.

On the scientific level, potential future work addresses the limitations of the proposed cognitive architecture regarding a *further* development of skills. For this purpose, two issues need to be addressed: goal selection and feature extension.

Currently, the selection of goal points is pre-defined, e.g. a given list of goal points in the workspace or a moving target object for interaction. Here, it would be useful to extend the proposed architecture by methods that self-generate goals [125], e.g. through artificial curiosity [160]. The ability to self-generate goal positions would imply the emergence of new skills based on the already available ones, like for example reaching to a goal while circumventing a physical obstacle lying in the trajectory planned by the PAS. Through goal resetting, subgoals can be devised resulting into to a new trajectory that is going around the obstacle. Note that this solely requires goal selection; the proposed PAS that takes the current goal as its input remains the same.

Feature extension would scale up the visual feature vector to contain more than the position of a particular feature of interest. For example, the feature vector could contain multiple points representing the shape of objects. By this extension, the robot would explore possibilities for advanced manipulation like stacking objects or inserting an object into slots, e.g. into a

matching aperture. The proposed PAS would facilitate a robust execution of such a new skill, since the EO-MTRNN is scalable in its spatial dimension to represent more (visual) features. Finally, the integration of the proposed episodic memory would be useful to trigger or to switch between procedural actions depending on the current environmental situation.
On the implementation level, potential future work would be to automatize the switching between the developmental stages of the architecture, as well as the switching between the different predictors inside the architecture. Technically, an option would be to transfer the implementation of the proposed methods to GPU. This would allow to scale up the spatial and temporal dimension of the EO-MTRNN, as well as the spatial dimension of the MLP, while significantly reducing the training time.

A. Appendix

Design Paradigms of Cognitive Architectures

Here, I summarize the design paradigms that currently exist in the field of cognitive architectures. Three main design paradigms exist that are also referred to as paradigms of cognition [210], [211], [208]: *cognitivist*, *emergent*, and *hybrid*.
See Appendix B for concrete examples of state-of-the-art cognitivist, emergent, and hybrid cognitive architectures.

A.1. Cognitivist Architecture Design

Cognitivist cognitive architectures are designed according to the paradigm of *cognitivism* [210]. Cognitivism has its origin in the 1950s and 1960s, when scientists and researchers tried to formally describe the human mind. At that time, activities such as walking or object recognition, which a human performs without conscious abstract thinking, were considered to be easy to reproduce artificially; hence, they were first neglected. The emphasis was on logical thinking and reasoning, since researchers considered these activities as the difficult ones. Thus, a computer should be able to reason and plan based on rule-based or knowledge-based algorithms. This was the beginning of the information processing or symbol manipulation approach to cognition, known as cognitivism, which can be seen as the historical origin of AI research. In Section 2.7.1, the first definition of cognition represents this point of view. According to [210], cognitivist systems are *physical symbol systems*. Newell and Simon [124] initially introduced the hypothesis of a physical symbol system. A physical symbol system contains two main abstract components. One component is a given set of symbolic structures. The other component is a set of processes that manipulate a symbolic input structure and produce a new symbolic output structure. The central assumption behind physical symbol systems is that any intelligent behaviour relies on abstract symbol manipulation. In general, the input is a given set of symbols, then the system performs operations on these symbols until it creates a symbolic structure representing a solution to a problem. In order to find a solution, the system uses the method of heuristic search [124]. Heuristic search starts with processing a given set of symbols and testing the result (new symbols). The system continues to generate new symbol structures and test them until a generated symbol structure matches a solution.
On the technical level, a physical symbol system can be realized by a *production system* that is also referred to as *rule-based system* [210], [208]. It is a system for executing condition-dependent actions/operations, relying on a given set of condition-action pairs, e.g. *if-then*-rules.
A cognitivist architecture might be, but does not necessarily have to be, embodied. This also

implies that it does not have to be connected to a perceptual system or to actuators. Applied to robotics, a cognitivist architecture is a symbol system with a predefined knowledge set, e.g. a world model, and data channels to sensors and actuators. On the technical level, cognitivist architectures consist of several functional modules with hardwired data connections. When the functional modules work together, they produce *coherent cognition* [3]. Note that many possibilities exist to implement a symbol system on the technical level.

A.1.1. Methods Implemented in Cognitivist Architectures

I list up typical methods that cognitivist architectures use (especially those focusing on perception in terms of vision). See the references for the implementation details:

- Fuzzy logic (e.g. fuzzy metric temporal Horn logic [119], [64]) and description logic (e.g. predicate logic [122]) represent knowledge about image sequences describing a dynamic scene such as an urban traffic scenario. Extracting knowledge about image sequences is a basic step for the construction of autonomous surveillance systems.

- Probabilistic frameworks / generative models [36], e.g. Bayesian networks and Hidden Markov models, learn to estimate the dynamics of a scene; for example, they learn to estimate trajectories of persons or objects within a scene.

- Situation graph trees [5], [119] provide conceptual knowledge that helps to generate textual descriptions of image sequences.

- Hierarchical task networks [5] provide a formal description for planning.

- Ontologies [112] connect specific image processing knowledge to domain knowledge. This connection to domain knowledge is important, since domain knowledge can be applied to various tasks, independent from the vision system [112].

- Semantic reasoning [144] fuses information from different signals to enable a robot to segment and recognize human activities.

Although these methods differ in details, many have in common that they consist of symbolic structures. These structures are directly interpretable by a human mind. They represent an initial amount of knowledge required for reasoning, e.g. predicate logic and fuzzy logic.

A.1.2. Drawbacks of Cognitivist Architectures

All implementations of symbol systems have in common that the symbols represent labelled entities in the world [30], e.g. objects, concepts, colours. This leads to symbol grounding problems because these symbols have more meaning to a human observer than to the system itself [30].

Moreover, the cognitivist approach can have some drawbacks when implementing it on humanoid robots operating in a real world. The robot's behaviour can be quite slow and suffer from long reaction times [35]. In addition, the internal symbolic representation that describes the world along with the necessary knowledge can often differ from the real world or real situations. In a real world, unforeseen environmental changes might happen, which were not taken into account by the designer of the architecture.

A.2. Emergent Architecture Design

In contrast to cognitivism, the approach of *emergent* cognitive architectures takes a different view on cognition. Instead of well-defined syntactic manipulation of symbols, the computational operation relies to a lesser or greater extent on processes of self-organization [210], [139]. In an emergent system, an internal symbol system (like the one in a cognitivist architecture) does not exist, a world model along with robot behaviour is not represented symbolically, so the system designer cannot access it or specify it at the outset. Pfeifer [137] claims that the symbol grounding problem is not an issue because the agent's behaviour is grounded in its sensory-motor coordination. Instead of symbolic data structures describing robot actions from the start on, the actions *emerge* dynamically as a result of the interaction between robot and environment.
Pfeifer et al. [139] describe this interaction between robot and environment, and they state one of the information-theoretic implications of embodiment:

> "(...) information structure does not exist before the interaction occurs but emerges only when the embodied system interacts with its surroundings."
> [139, p. 1089]

While interacting with the real world, more complex actions or behaviour can emerge based on simpler ones. This principle of emergence implies a close sensory-motor coupling and explains why a physical body is necessary for the emergent approach. Thus, emergence implies embodiment [210]. The agent has to be embodied in order to sense its environment through a rich perceptual interface and to act in the environment. Otherwise, cognition is not possible according to its second and third definition (Section 2.7.1).
In contrast to the cognitivist approach, the emergent approach provides fast behavioural responses [111] and allows a close coupling between internal system states and the real world. Emergent architectures split up into three categories [210], [208]: *connectionist* architectures, *dynamical* architectures, and *enactive* architectures.
Based on [29], [30], [7], [110], [111], [140], [138], [139], [210], [206], [211], the main properties of emergent systems are:

- More direct coupling of perception to action
- Distributed processing, self-organization
- Interaction with the real world (implies embodiment)

- Behaviour emerges as the result of the interplay between the robot's control mechanism, its morphology, and the environment

- Complex behaviour can emerge from several simple behaviours

A.2.1. Methods Implemented in Emergent Architectures

I list up typical methods that emergent architectures use, together with the works describing or implementing them:

- Finite state machines [29], [30] realize a system of distributed agents that can run asynchronously. A central controller is not needed.

- Potential fields [7] allow the concurrent combination of different motor schemas used for mobile robot navigation.

- Gradient fields [136] can be used for path or route planning, since they bias the robot's decision which direction to take.

- Activation/inhibition dynamics [110] can implement emergent action selection.

- Markov decision processes [218] are used to mathematically model a developmental architecture for robots.

- Dynamical systems theory [197], [162], [58] describes behaviour and learning. A behaviour is represented by an attractor state of a dynamical system. Instabilities of the system dynamics lead to a change of behaviour and effect a form of learning.

- Concepts of self-organization [137], [139], which are closely related to dynamical systems, replace conventional top-down control by the tight coupling of (distributed) control mechanisms, morphology, and environment. Self-organization can lead to unforeseen, emergent behaviour.

- Artificial neural networks, e.g. [171], [2], [20], [133], [39], [94], [230], realize multiple capabilities such as learning, pattern classification, prediction, and simple robot behaviour. A benefit of many neural networks is their ability to generalize over the training data, for example in [133], [39].

- Multi-modal sensory-motor mapping [61], [8], [181] enables a humanoid robot to acquire a representation of its body (*body schema* or *body image*) and to acquire simple behaviour inspired by findings in developmental psychology. Body schema also plays a key role in tool use [118].

A.3. Comparison between Cognitivist and Emergent Architectures

Based on [210], [208], [111], I briefly compare the strengths and weaknesses of each cognitivist and emergent design approach in Table 29.

Design approach	**Advantages**	**Disadvantages**
Cognitivist	More advanced capabilities than emergent models, given that the right knowledge is provided	Difficulties with noisy data and dynamic environments Problematic modelling of generalization capabilities Knowledge provided by the system designer, not acquired by the system itself
Emergent	More robust, less brittle than cognitivist models Better adaptivity to environmental changes Fast behavioural responses to disturbances	Scientific methods and robot capabilities limited at present

Table 29 Comparison between cognitivist and emergent architectures, based on [210], [208], [111].

A.4. Hybrid Architecture Design

Hybrid architectures combine the cognitivist design approach with the emergent design approach in order to compensate the weaknesses of each other. An important characteristic feature of a hybrid architecture is that it uses symbolic representations that are constructed by the architecture itself by interacting with the environment [210], [208]. The utilization of symbolic representations is a characteristic feature of the cognitivist approach, while the self-construction of new symbolic representations (i.e. new knowledge) through environmental interaction is a characteristic feature of the emergent approach.

B. Appendix

Examples of Cognitive Architectures

Here, I briefly list some examples of state-of-the-art cognitive architectures, each example with the reference to its original paper.

B.1. Cognitivist Architectures

Some prominent *cognitivist* cognitive architectures are:

- *Soar* [98], [97] is a production system with a set of different memories to solve abstract tasks. A given task is represented within a problem space. The *Soar* architecture uses rules and knowledge to work out a solution in form of state transitions within the problem space, leading to a desired (goal) state. *Soar* can also learn new rules. *Soar* is aimed at a unified theory of cognition.

- *EPIC* [87] consists of different sensory-motor processors (e.g. a visual processor for visual input, an auditory processor for auditory input, a tactile processor for tactile input, etc.), a working memory, and a production system. The sensory-motor processors model the time of transfer of stimulus representations to the working memory. *EPIC* does not support learning. It is applied to model human performance when interacting with machines and their peripheries, e.g. computer keyboards and monitors.

- *ACT-R* [3] contains a production system that processes visuo-motor data together with memory content and a goal state. Within *ACT-R*, learning consists of the tuning of parameters that influence the choice of production rules. Like *Soar* [98], [97], *ACT-R* is aimed at a unified theory of cognition.

- *ADAPT* [21] works with a hierarchy of perceptual and planning schemas. Schemas are representations of perception and action.

- *ICARUS* [43] uses rules and various types of memory to solve tasks in a simulated environment. Memory content is split into concepts and skills. On the one hand, concept memory encodes knowledge about object categories including their relations. On the other hand, skill memory encodes knowledge about ways to act to achieve goals. In a simulated environment of a virtual city, *ICARUS* controls an agent that drives to target locations.

- *GLAIR* [166] consists of a sensory-actuator layer for low-level control, a high-level knowledge layer for reasoning and planning, and a perceptuo-motor layer for connecting the former two layers. The main capability of *GLAIR* is reasoning and language comprehension.

B.2. Emergent Architectures

Some prominent *emergent* cognitive architectures are:

- *Autonomous agent robotics* (*AAR*), also called *behaviour-based robotics* based on the *subsumption architecture* [29], consists of several layers of control realized by finite state machines along with suppression and inhibition mechanisms. Instead of establishing functional modules, the design principle focuses on establishing a hierarchy of behaviour. In [29], the architecture controls a mobile robot.

- *Self-directed anticipative learning* (*SDAL*) [48] is a theoretical approach to construct dynamical embodied systems with adaptiveness and anticipation as main features. However, a computational model realizing the approach is missing.

- The *self-aware self-effecting* (*SASE*) cognitive architecture [214], [218] consists of internal imaginary sensors and effectors in addition to the external physical ones. The internal sensors and effectors work in an internal environment that is the representation of the system's "brain". *SASE* is an architecture designed for mental development [217] through the interaction between the robot and its environment. The architecture [218] is realized by a system of Markov decision processes.

- *Neuro-mimetic robotic brain-based devices* (*BBDs*) named *Darwin* [91], [92] emulate the structure and organization of the brain, with the main purpose to test theories about the brain. A *BBD* is embedded into a robot to consider operation in the real world. It is stated that a *BBD* categorizes perceptions in an online manner without *a priori* knowledge. Further capabilities include cross-modal sensory-motor associations and predictive motor control tested on a mobile robot.

- The *Global Workspace cognitive architecture* [164] contains an internal sensory-motor loop that realizes prediction capability. See Section 2.9.3 for more details.

- The *embodied cognitive-affective architecture* [235] provides a model for the integration of emotion into cognitive processes. It is stated that emotion is grounded in the automatic regulation of physiological functions, i.e. *homeostasis*.

- The *iCub cognitive architecture* [209], [211], [206], [207] consists of a network of functional modules such as attention selection, episodic and procedural memory, locomotion, reaching and grasping control, affective state (providing motives for actions), and action selection. The robot's behaviour emerges as a result of the interaction of these sub-systems. The architecture is designed for the integration of basic robot skills, e.g. hand-eye coordination. The architecture can run an internal simulation of sensory-motor states. The simulation outcome can influence the motives guiding the action selection. The *iCub cognitive architecture* is an example for motive-driven action selection, see Section 2.8.3.

B.3. Hybrid Architectures

Some prominent *hybrid* cognitive architectures are:

- The *Cerebus* architecture [74] combines behaviour-based systems (like *AAR*) with a symbolic reasoning system using predicate logic. *Cerebus* is implemented on a mobile robot that can reason about its own behaviours (e.g. follow a human) to a certain limited extend.

- *Kismet* [26], [27] is both a cognitive architecture and the name of an expressive robot head. The main parts of the architecture are an emotion system, a drive system, and a behaviour system. *Kismet* is aimed at investigating the role of emotion and facial expressions in social interactions between humans and robots.

- The *theory of mind* architecture [157] for the humanoid robot Cog [32] aims at establishing non-linguistic social skills. The architecture can direct the robot's attention to eyes and faces, and distinguish between animate and inanimate motion.

- A *humanoid robot cognitive architecture* [35] is proposed that consists of three layers, i.e. a low-level, a mid-level, and a top-level layer. The low-level layer contains fast reactive modules of perception and task execution, whereas the high-level contains planning and learning modules working on symbolic level together with a knowledge database. The modules interact through an "active model" that is a kind of short-term memory. The architecture is used to investigate problems in the domain of service robots operating in households.

- *CLARION* [184] works with four main types of internal representations. Representations are split into action-centred and non-action-centred types. Each of these types further contains explicit (symbolic) and implicit (sub-symbolic, i.e. neural network parametric) representations. The architecture combines symbolic processing (e.g. reasoning) with connectionist processing. The architecture is used for simulations of psychological experiments.

- The *learning intelligent distribution agent* (*LIDA*) [13] contains a network of different memory modules (episodic, declarative, procedural). *LIDA* uses a competition process of the global workspace theory [12] in order to implement a kind of attention selection. Learning happens in the phase of action selection when memory contents are updated as a result of the competition process.

- *PACO-PLUS* [141] is a three-layered architecture with a strong focus on learning. The architecture supports learning by self-exploration and learning by imitation. When interacting with objects, the robot can observe the consequences of its own action, i.e. the new perceptual state as the result of its action. Thus, *PACO-PLUS* can learn object-action complexes [228].

C. Appendix

Architectures Emphasizing Memory

Langley et al. [100], [99] highlight the importance of memories in cognitive architectures (see Section 2.7.2).
An example architecture is the *Interaction History Architecture (IHA)* [114] that relies on information distance between sensory-motor flows over a finite time horizon, i.e. operationalized experiences.
A further development is the *Extended Interaction History Architecture (EIHA)* [33]. Its short-term memory contains an interaction history between a human and the robot. *EIHA* can acquire a peek-a-boo and drumming behaviour on the iCub robot. Switching between the behaviours is seen as a response to social engagement feedback from a human interacting with the robot.
Interaction histories combined with information theoretic methods also facilitate bottom-up language acquisition without prior representations [121], [156].

Bibliography

[1] R. A. Adams, S. Shipp, and K. J. Friston. Predictions not commands: Active inference in the motor system. *Brain Structure and Function*, 218(3):611–643, 2013. (Cited on pages 22 and 23.)

[2] I. Aleksander. Neural systems engineering: Towards a unified design discipline? *Computing & Control Engineering Journal*, 1(6):259–265, 1990. (Cited on pages 40 and 190.)

[3] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, 2004. (Cited on pages 188 and 193.)

[4] R. L. Anderson. Recent advances in finding best operating conditions. *Journal of the American Statistical Association*, 48(264):789–798, 1953. (Cited on page 52.)

[5] M. Arens and H.-H. Nagel. Representation of behavioral knowledge for planning and plan-recognition in a cognitive vision system. In M. Jarke, G. Lakemeyer, and J. Koehler, editors, *KI 2002: Advances in Artificial Intelligence*, volume 2479 of *Lecture Notes in Artificial Intelligence*, pages 268–282. Berlin, Germany: Springer, 2002. (Cited on page 188.)

[6] H. Arie, T. Arakaki, S. Sugano, and J. Tani. Imitating others by composition of primitive actions: A neuro-dynamic model. *Robotics and Autonomous Systems*, 60(5):729–741, 2012. (Cited on pages 18, 23, 25, 37, 40, 41, 42, 45, 48, and 49.)

[7] R. C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. *Robotics and Autonomous Systems*, 6(1–2):105–122, 1990. (Cited on pages 189 and 190.)

[8] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida. Cognitive developmental robotics: A survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):12–34, 2009. (Cited on pages 16, 151, and 190.)

[9] M. Asada, K. F. MacDorman, H. Ishiguro, and Y. Kuniyoshi. Cognitive developmental robotics as a new paradigm for the design of humanoid robots. *Robotics and Autonomous Systems*, 37(2–3):185–193, 2001. (Cited on page 16.)

[10] A. J. Ayer. *Language, Truth and Logic*. New York, NY, USA: Dover Publications, Inc., 1952. (Cited on pages 27 and 154.)

[11] B. J. Baars. *A Cognitive Theory of Consciousness*. New York, NY, USA: Cambridge University Press, 1988. (Cited on page 39.)

[12] B. J. Baars. Global workspace theory of consciousness: Toward a cognitive neuroscience of human experience. In S. Laureys, editor, *The Boundaries of Consciousness: Neurobiology and Neuropathology*, volume 150 of *Progress in Brain Research*, pages 45–53. Amsterdam, Netherlands: Elsevier, 2005. (Cited on pages 39 and 195.)

[13] B. J. Baars and S. Franklin. Consciousness is computational: The LIDA model of global workspace theory. *International Journal of Machine Consciousness*, 1(1):23–32, 2009. (Cited on page 195.)

[14] D. Badre. Cognitive control, hierarchy, and the rostro-caudal organization of the frontal lobes. *Trends in Cognitive Sciences*, 12(5):193–200, 2008. (Cited on pages 22 and 148.)

[15] D. Badre and M. D'Esposito. Functional magnetic resonance imaging evidence for a hierarchical organization of the prefrontal cortex. *Journal of Cognitive Neuroscience*, 19(12):2082–2099, 2007. (Cited on page 23.)

[16] D. Badre and M. D'Esposito. Is the rostro-caudal axis of the frontal lobe hierarchical? *Nature Reviews Neuroscience*, 10(9):659–669, 2009. (Cited on pages 22, 23, 49, 116, 120, and 148.)

[17] A. Baranes and P.-Y. Oudeyer. R-IAC: Robust intrinsically motivated exploration and active learning. *IEEE Transactions on Autonomous Mental Development*, 1(3):155–169, 2009. (Cited on pages 20 and 45.)

[18] A. Baranes and P.-Y. Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013. (Cited on pages 19, 21, 43, 45, and 102.)

[19] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993. (Cited on page 53.)

[20] R. D. Beer, H. J. Chiel, and L. S. Sterling. A biological perspective on autonomous agent design. *Robotics and Autonomous Systems*, 6(1–2):169–186, 1990. (Cited on page 190.)

[21] D. P. Benjamin, D. Lyons, and D. Lonsdale. ADAPT: A cognitive architecture for robotics. In *Proceedings of the International Conference on Cognitive Modeling*, Pittsburgh, PA, USA, pages 337–338, 2004. (Cited on page 193.)

[22] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. (Cited on page 52.)

[23] N. A. Bernstein. *The Co-ordination and Regulation of Movements*. Oxford, UK: Pergamon Press Ltd., 1967. (Cited on page 102.)

[24] A. Berthoz. *The Brain's Sense of Movement*. Cambridge, MA, USA: Harvard University Press, 2000. (Cited on page 31.)

[25] M. H. Bickhard. Autonomy, function, and representation. *Artificial Intelligence, Special Issue on Communication and Cognition*, 17(3–4):111–131, 2000. (Cited on page 29.)

[26] C. L. Breazeal. *Designing Sociable Robots*. Cambridge, MA, USA: MIT Press, 2002. (Cited on page 195.)

[27] C. L. Breazeal. Emotion and sociable humanoid robots. *International Journal of Human-Computer Studies*, 59(1–2):119–155, 2003. (Cited on page 195.)

[28] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006. (Cited on pages xxi, 52, 60, 62, 63, 75, 76, 87, 88, and 182.)

[29] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986. (Cited on pages 16, 29, 33, 34, 35, 38, 43, 45, 189, 190, and 194.)

[30] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6(1–2):3–15, 1990. (Cited on pages 16, 188, 189, and 190.)

[31] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1–3):139–159, 1991. (Cited on page 16.)

[32] R. A. Brooks, C. Breazeal, M. Marjanović, B. Scassellati, and M. M. Williamson. The Cog project: Building a humanoid robot. In C. L. Nehaniv, editor, *Computation for Metaphors, Analogy, and Agents*, volume 1562 of *Lecture Notes in Computer Science*, pages 52–87. Berlin, Germany: Springer, 1999. (Cited on page 195.)

[33] F. Broz, C. L. Nehaniv, H. Kose-Bagci, and K. Dautenhahn. Interaction histories and short term memory: Enactive development of turn-taking behaviors in a childlike humanoid robot. *arXiv:1202.5600v1*, February 25th, 2012. (Cited on page 197.)

[34] W. Burger*, E. Wieser*, E. Dean-Leon, and G. Cheng. A scalable method for multi-stage developmental learning for reaching. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Lisbon, Portugal, pages 60–65, 2017.
*W. Burger and E. Wieser had an equal contribution to the paper. (Cited on pages 92, 115, 125, 139, 141, 142, 143, 158, 161, and 167.)

[35] C. Burghart, R. Mikut, R. Stiefelhagen, T. Asfour, H. Holzapfel, P. Steinhaus, and R. Dillmann. A cognitive architecture for a humanoid robot: A first approach. In *Proceedings of the Fifth IEEE International Conference on Humanoid Robots*, Tsukuba, Japan, pages 357–362, 2005. (Cited on pages 188 and 195.)

[36] H. Buxton. Learning and understanding dynamic scene activity: A review. *Image and Vision Computing*, 21(1):125–136, 2003. (Cited on page 188.)

[37] D. Caligiore, M. Mirolli, D. Parisi, and G. Baldassarre. A bioinspired hierarchical reinforcement learning architecture for modeling learning of multiple skills with continuous states and actions. In B. Johansson, E. Sahin, and C. Balkenius, editors, *Proceedings of the Tenth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, volume 149, pages 27–34. Lund, Sweden: Lund University Cognitive Studies, 2010. (Cited on page 27.)

[38] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 37(2):286–298, 2007. (Cited on page 22.)

[39] T. Chaminade, E. Oztop, G. Cheng, and M. Kawato. From self-observation to imitation: Visuomotor association on a robotic hand. *Brain Research Bulletin*, 75(6):775–784, 2008. (Cited on pages 164, 165, and 190.)

[40] Y. Chen, S. Murata, H. Arie, T. Ogata, J. Tani, and S. Sugano. Emergence of interactive behaviors between two robots by prediction error minimization mechanism. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Cergy-Pontoise, Paris, France, pages 302–307, 2016. (Cited on page 25.)

[41] G. Cheng and Y. Kuniyoshi. Complex continuous meaningful humanoid interaction: A multi sensory-cue based approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, USA, volume 3, pages 2235–2242, 2000. (Cited on pages 35, 38, 41, 42, 43, 122, and 152.)

[42] G. Cheng, A. Nagakubo, and Y. Kuniyoshi. Continuous humanoid interaction: An integrated perspective — gaining adaptivity, redundancy, flexibility — in one. *Robotics and Autonomous Systems*, 37(2–3):161–183, 2001. (Cited on pages 26, 35, 38, 41, 42, and 43.)

[43] D. Choi, M. Kaufman, P. Langley, N. Nejati, and D. Shapiro. An architecture for persistent reactive behavior. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (IEEE Computer Society)*, New York, NY, USA, volume 2, pages 986–993, 2004. (Cited on page 193.)

[44] M. Choi and J. Tani. Predictive coding for dynamic visual processing: Development of functional hierarchy in a multiple spatiotemporal scales RNN model. *Neural Computation*, 30(1):237–270, 2018. (Cited on page 25.)

[45] N. Chomsky. *Language and Mind*. New York, NY, USA: Cambridge University Press, 3rd edition, 2006. (Cited on pages 4 and 93.)

[46] P. A. Chouinard. Different roles of PMv and PMd during object lifting. *The Journal of Neuroscience*, 26(24):6397–6398, 2006. (Cited on page 120.)

[47] P. A. Chouinard and T. Paus. The primary motor and premotor areas of the human cerebral cortex. *The Neuroscientist*, 12(2):143–152, 2006. (Cited on page 120.)

[48] W. D. Christensen and C. A. Hooker. An interactivist-constructivist approach to intelligence: Self-directed anticipative learning. *Philosophical Psychology*, 13(1):5–45, 2000. (Cited on page 194.)

[49] W. D. Christensen and C. A. Hooker. Representation and the meaning of life. In H. Clapin, P. Staines, and P. Slezak, editors, *Representation in Mind: New Approaches to Mental Representation*, pages 41–70. New York, NY, USA: Elsevier, 2004. (Cited on page 34.)

[50] A. Clark. Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and Brain Sciences*, 36(3):181–204, 2013. (Cited on page 22.)

[51] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. (Cited on page 53.)

[52] E. Dean-Leon, B. Pierce, F. Bergner, P. Mittendorfer, K. Ramirez-Amaro, W. Burger, and G. Cheng. TOMM: Tactile omnidirectional mobile manipulator. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Singapore, Singapore, pages 2441–2447, 2017. (Cited on pages 8, 42, 91, 115, 139, 151, and 167.)

[53] Y. Demiris and A. Dearden. From motor babbling to hierarchical learning by imitation: A robot developmental pathway. In L. Berthouze, F. Kaplan, H. Kozima, H. Yano, J. Konczak, G. Metta, J. Nadel, G. Sandini, G. Stojanov, and C. Balkenius, editors, *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, volume 123, pages 31–37. Lund, Sweden: Lund University Cognitive Studies, 2005. (Cited on pages 26, 39, 41, and 42.)

[54] Y. Demiris and B. Khadhouri. Hierarchical attentive multiple models for execution and recognition of actions. *Robotics and Autonomous Systems*, 54(5):361–369, 2006. (Cited on pages 39, 41, and 42.)

[55] K. Doya. What are the computations of the cerebellum, the basal ganglia and the cerebral cortex? *Neural Networks*, 12(7–8):961–974, 1999. (Cited on pages 5, 17, and 18.)

[56] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the IEEE Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995. (Cited on page 52.)

[57] J. L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. (Cited on pages 24 and 48.)

[58] W. Erlhagen and E. Bicho. The dynamic neural field approach to cognitive robotics. *Journal of Neural Engineering*, 3(3):R36–R54, 2006. (Cited on page 190.)

[59] K. Friston. Hierarchical models in the brain. *PLoS Computational Biology*, 4(11):e1000211, 2008. (Cited on pages 2, 22, 23, 25, 44, and 154.)

[60] K. Friston. The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, 11(2):127–138, 2010. (Cited on pages 2, 7, 22, 23, 44, 45, 48, 49, 92, and 154.)

[61] S. Fuke, M. Ogino, and M. Asada. Body image constructed from motor and tactile images with visual information. *International Journal of Humanoid Robotics*, 4(2):347–364, 2007. (Cited on pages 19 and 190.)

[62] J. M. Fuster. The prefrontal cortex — an update: Time is of the essence. *Neuron*, 30(2):319–333, 2001. (Cited on page 49.)

[63] D. George and J. Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Computational Biology*, 5(10):e1000532, 2009. (Cited on page 18.)

[64] R. Gerber, H.-H. Nagel, and H. Schreiber. Deriving textual descriptions of road traffic queues from video sequences. In F. van Harmelen, editor, *ECAI'02: Proceedings of the 15th European Conference on Artificial Intelligence*, pages 736–740. Amsterdam, Netherlands: IOS Press, 2002. (Cited on page 188.)

[65] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000. (Cited on page 48.)

[66] E. C. Goldfield. *Emergent Forms: Origins and Early Development of Human Action and Perception*. New York, NY, USA: Oxford University Press, 1995. (Cited on page 102.)

[67] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. `http://www.deeplearningbook.org`. (Cited on page 58.)

[68] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017. (Cited on page 52.)

[69] F. Guerin, N. Krüger, and D. Kraft. A survey of the ontogeny of tool use: From sensorimotor experience to planning. *IEEE Transactions on Autonomous Mental Development*, 5(1):18–45, 2013. (Cited on page 28.)

[70] M. Haruno, D. M. Wolpert, and M. Kawato. MOSAIC model for sensorimotor learning and control. *Neural Computation*, 13(10):2201–2220, 2001. (Cited on pages 39, 41, and 42.)

[71] W. Hinoshita, H. Arie, J. Tani, T. Ogata, and H. G. Okuno. Recognition and generation of sentences through self-organizing linguistic hierarchy using MTRNN. In N. García-Pedrajas, F. Herrera, C. Fyfe, J. M. Benítez, and M. Ali, editors, *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, volume 6098 of *Lecture Notes in Artificial Intelligence*, pages 42–51. Berlin, Germany: Springer, 2010. (Cited on page 48.)

[72] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. (Cited on page 48.)

[73] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982. (Cited on page 164.)

[74] I. Horswill. Tagged behavior-based systems: Integrating cognition with embodied activity. *IEEE Intelligent Systems*, 16(5):30–37, 2001. (Cited on page 195.)

[75] H. Jaeger. The “echo state” approach to analysing and training recurrent neural networks. Technical report, TR, GMD Report 148, German National Research Center for Information Technology, Sankt Augustin, Germany, 2001. (Cited on page 48.)

[76] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004. (Cited on page 48.)

[77] S. Jeong, H. Arie, M. Lee, and J. Tani. Neuro-robotics study on integrative learning of proactive visual attention and motor behaviors. *Cognitive Neurodynamics*, 6(1):43–59, 2012. (Cited on page 48.)

[78] M. H. Johnson. *Developmental Cognitive Neuroscience*. Hoboken, NJ, USA: John Wiley & Sons, 1997. (Cited on pages 2 and 154.)

[79] M. I. Jordan. Serial order: A parallel distributed processing approach. Technical report, TR - ICS Report 8604, Institute for Cognitive Science, University of California, San Diego, CA, USA, 1986. (Cited on pages 24 and 48.)

[80] M. I. Jordan. Serial order: A parallel distributed processing approach. In J. W. Donahoe and V. P. Dorsel, editors, *Neural-Network Models of Cognition: Biobehavioral Foundations*, volume 121 of *Advances in Psychology*, pages 471–495. Amsterdam, Netherlands: Elsevier, 1997. (Cited on pages 24 and 48.)

[81] M. I. Jordan and D. E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16(3):307–354, 1992. (Cited on page 18.)

[82] M. Jung, J. Hwang, and J. Tani. Self-organization of spatio-temporal hierarchy via learning of dynamic visual image patterns on action sequences. *PLoS One*, 10(7):e0131214, 2015. (Cited on page 50.)

[83] I. Kajić, G. Schillaci, S. Bodiroža, and V. V. Hafner. Learning hand-eye coordination for a humanoid robot using SOMs. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction*, Bielefeld, Germany, pages 192–193, 2014. (Cited on pages 8, 92, and 102.)

[84] R. Kanai, Y. Komura, S. Shipp, and K. Friston. Cerebral hierarchies: Predictive processing, precision and the pulvinar. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 370(1668):20140169, 2015. (Cited on page 22.)

[85] S. Karsoliya. Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture. *International Journal of Engineering Trends and Technology*, 3(6):714–717, 2012. (Cited on page 170.)

[86] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, Perth, WA, Australia, volume 4, pages 1942–1948, 1995. (Cited on page 52.)

[87] D. E. Kieras and D. E. Meyer. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*, 12(4):391–438, 1997. (Cited on page 193.)

[88] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982. (Cited on pages 6, 20, 49, and 53.)

[89] T. Kohonen. *Self-Organization and Associative Memory*. Springer Series in Information Sciences. Berlin, Germany: Springer, 3rd edition, 1989. (Cited on pages 6, 20, 49, and 53.)

[90] J. Koster-Hale and R. Saxe. Theory of mind: A neural prediction problem. *Neuron*, 79(5):836–848, 2013. (Cited on page 22.)

[91] J. L. Krichmar and G. M. Edelman. Brain-based devices for the study of nervous systems and the development of intelligent machines. *Artificial Life*, 11(1–2):63–77, 2005. (Cited on page 194.)

[92] J. L. Krichmar and G. M. Edelman. Design principles and constraints underlying the construction of brain-based devices. In M. Ishikawa, K. Doya, H. Miyamoto, and T. Yamakawa, editors, *Neural Information Processing. ICONIP 2007*, volume 4985 of *Lecture Notes in Computer Science*, pages 157–166. Berlin, Germany: Springer, 2008. (Cited on page 194.)

[93] N. Krüger, M. Popovic, L. Bodenhagen, D. Kraft, and F. Guerin. Grasp learning by means of developing sensorimotor schemas and generic world knowledge. In *Proceedings of the Convention on Artificial Intelligence and Simulation of Behaviour*, pages 23–31, 2011. (Cited on page 28.)

[94] Y. Kuniyoshi and S. Sangawa. Early motor development from partially ordered neural-body dynamics: Experiments with a cortico-spinal-musculo-skeletal model. *Biological Cybernetics*, 95(6):589–605, 2006. (Cited on page 190.)

[95] Y. Kuniyoshi, Y. Yorozu, M. Inaba, and H. Inoue. From visuo-motor self learning to early imitation — a neural architecture for humanoid learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, volume 3, pages 3132–3139, 2003. (Cited on pages 22 and 178.)

[96] Y. Kuniyoshi, Y. Yorozu, Y. Ohmura, K. Terada, T. Otani, A. Nagakubo, and T. Yamamoto. From humanoid embodiment to theory of mind. In F. Iida, R. Pfeifer, L. Steels, and Y. Kuniyoshi, editors, *Embodied Artificial Intelligence*, volume 3139 of *Lecture Notes in Artificial Intelligence*, pages 202–218. Berlin, Germany: Springer, 2004. (Cited on page 102.)

[97] J. E. Laird. *The Soar Cognitive Architecture*. Cambridge, MA, USA: MIT Press, 2012. (Cited on page 193.)

[98] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987. (Cited on page 193.)

[99] P. Langley. Cognitive architectures and general intelligent systems. *AI Magazine*, 27(2):33–44, 2006. (Cited on pages 31 and 197.)

[100] P. Langley, J. E. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009. (Cited on pages 31, 32, 33, and 197.)

[101] P. Lanillos, E. Dean-Leon, and G. Cheng. Yielding self-perception in robots through sensorimotor contingencies. *IEEE Transactions on Cognitive and Developmental Systems*, 9(2):100–112, 2017. (Cited on page 19.)

[102] J. Law, M. Lee, M. Hülse, and A. Tomassetti. The infant development timeline and its application to robot shaping. *Adaptive Behavior*, 19(5):335–358, 2011. (Cited on page 40.)

[103] J. Law, P. Shaw, K. Earland, M. Sheldon, and M. Lee. A psychology based approach for longitudinal development in cognitive robotics. *Frontiers in Neurorobotics*, 8(1):1–19, 2014. (Cited on pages 40, 41, 42, 44, and 45.)

[104] J. Law, P. Shaw, and M. Lee. A biologically constrained architecture for developmental learning of eye–head gaze control on a humanoid robot. *Autonomous Robots*, 35(1):77–92, 2013. (Cited on page 40.)

[105] C.-Y. Lee and X. Yao. Evolutionary programming using mutations based on the Lévy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8(1):1–13, 2004. (Cited on page 52.)

[106] M. H. Lee, Q. Meng, and F. Chao. Developmental learning for autonomous robots. *Robotics and Autonomous Systems*, 55(9):750–759, 2007. (Cited on page 40.)

[107] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing*, 9(6):448–462, 2005. (Cited on page 52.)

[108] J. J. Lockman. Perceptuomotor coordination in infancy. In C.-A. Hauert, editor, *Developmental Psychology: Cognitive, Perceptuo-Motor and Neuropsychological Perspectives*, volume 64 of *Advances in Psychology*, pages 85–111. Amsterdam, Netherlands: Elsevier, 1990. (Cited on pages 2, 32, and 151.)

[109] M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: A survey. *Connection Science*, 15(4):151–190, 2003. (Cited on pages 4, 16, 26, 27, 33, 93, and 102.)

[110] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6(1–2):49–70, 1990. (Cited on pages 16, 33, 34, 35, 38, 43, 45, 189, and 190.)

[111] P. Maes. *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. Cambridge, MA, USA: MIT Press, 1990. (Cited on pages xxii, 16, 33, 189, and 191.)

[112] N. Maillot, M. Thonnat, and A. Boucher. Towards ontology-based cognitive vision. *Machine Vision and Applications*, 16(1):33–40, 2004. (Cited on page 188.)

[113] H. R. Maturana and F. J. Varela. *The Tree of Knowledge: The Biological Roots of Human Understanding*. Boston, MA, USA: New Science Library / Shambhala Publications, 1987. (Cited on pages 29 and 43.)

[114] N. A. Mirza, C. L. Nehaniv, K. Dautenhahn, and R. te Boekhorst. Developing social action capabilities in a humanoid robot using an interaction history architecture. In *Proceedings of the IEEE International Conference on Humanoid Robots*, Daejeon, Korea, pages 609–616, 2008. (Cited on page 197.)

[115] P. Mittendorfer, E. Yoshida, and G. Cheng. Realizing whole-body tactile interactions with a self-organizing, multi-modal artificial skin on a humanoid robot. *Advanced Robotics*, 29(1):51–67, 2015. (Cited on page 19.)

[116] M. Morita. Memory and learning of sequential patterns by nonmonotone neural networks. *Neural Networks*, 9(8):1477–1489, 1996. (Cited on page 24.)

[117] H. G. Musmann, P. Pirsch, and H.-J. Grallert. Advances in picture coding. *Proceedings of the IEEE*, 73(4):523–548, 1985. (Cited on page 22.)

[118] C. Nabeshima, M. Lungarella, and Y. Kuniyoshi. Timing-based model of body schema adaptation and its role in perception and tool use: A robot case study. In *Proceedings of the Fourth IEEE International Conference on Development and Learning*, Osaka, Japan, pages 7–12, 2005. (Cited on page 190.)

[119] H.-H. Nagel. Steps toward a cognitive vision system. *AI Magazine*, 25(2):31–50, 2004. (Cited on page 188.)

[120] J. Nassour, V. Hugel, F. B. Ouezdou, and G. Cheng. Qualitative adaptive reward learning with success failure maps: Applied to humanoid robot walking. *IEEE Transactions on Neural Networks and Learning Systems*, 24(1):81–93, 2013. (Cited on pages 20, 21, and 27.)

[121] C. L. Nehaniv, F. Förster, J. Saunders, F. Broz, E. Antonova, H. Köse, C. Lyon, H. Lehmann, Y. Sato, and K. Dautenhahn. Interaction and experience in enactive intelligence and humanoid robotics. In *Proceedings of the IEEE Symposium on Artificial Life*, Singapore, Singapore, pages 148–155, 2013. (Cited on page 197.)

[122] B. Neumann and R. Möller. On scene interpretation with description logics. *Image and Vision Computing*, 26(1):82–101, 2008. (Cited on page 188.)

[123] A. Newell. *Unified Theories of Cognition*. Cambridge, MA, USA: Harvard University Press, 1994. (Cited on page 30.)

[124] A. Newell and H. A. Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the Association for Computing Machinery*, 19(3):113–126, Tenth Turing Award Lecture, 1976. (Cited on pages 16 and 187.)

[125] H. Ngo, M. Luciw, A. Förster, and J. Schmidhuber. Confidence-based progress-driven self-generated goals for skill acquisition in developmental robots. *Frontiers in Psychology*, 4(833):1–19, 2013. (Cited on page 185.)

[126] R. Nishimoto, J. Namikawa, and J. Tani. Learning multiple goal-directed actions through self-organization of a dynamic neural network model: A humanoid robot experiment. *Adaptive Behavior*, 16(2–3):166–181, 2008. (Cited on pages 40, 41, 42, 44, 45, 48, and 183.)

[127] R. Nishimoto and J. Tani. Learning to generate combinatorial action sequences utilizing the initial sensitivity of deterministic dynamical systems. *Neural Networks*, 17(7):925–933, 2004. (Cited on page 24.)

[128] K. Noda, K. Kawamoto, T. Hasuo, and K. Sabe. A generative model for developmental understanding of visuomotor experience. In *Proceedings of the IEEE International Conference on Development and Learning*, Frankfurt am Main, Germany, volume 2, pages 1–7, 2011. (Cited on pages 19 and 28.)

[129] Open Source Robotics Foundation. ROS — Robot Operating System, [Accessed February 8th, 2019]. Available at URL `https://www.ros.org`. (Cited on page 94.)

[130] OpenCV Team. OpenCV — Open Source Computer Vision Library, [Accessed February 8th, 2019]. Available at URL `https://opencv.org`. (Cited on page 94.)

[131] P.-Y. Oudeyer, A. Baranes, and F. Kaplan. Intrinsically motivated learning of real-world sensorimotor skills with developmental constraints. In G. Baldassarre and M. Mirolli, editors, *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 303–365. Berlin, Germany: Springer, 2013. (Cited on page 26.)

[132] 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. (Cited on pages 20 and 45.)

[133] E. Oztop, T. Chaminade, G. Cheng, and M. Kawato. Imitation bootstrapping: Experiments on a robotic hand. In *Proceedings of the IEEE International Conference on Humanoid Robots*, Tsukuba, Japan, pages 189–195, 2005. (Cited on pages 164, 165, and 190.)

[134] F. S. Panchal and M. Panchal. Review on methods of selecting number of hidden nodes in artificial neural network. *International Journal of Computer Science and Mobile Computing*, 3(11):455–464, 2014. (Cited on page 170.)

[135] J.-C. Park, J. H. Lim, H. Choi, and D.-S. Kim. Predictive coding strategies for developmental neurorobotics. *Frontiers in Psychology*, 3(134):1–10, 2012. (Cited on page 25.)

[136] D. W. Payton. Internalized plans: A representation for action resources. *Robotics and Autonomous Systems*, 6(1–2):89–103, 1990. (Cited on page 190.)

[137] R. Pfeifer. Building “Fungus Eaters”: Design principles of autonomous agents. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, *From Animals To Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 3–12. Cambridge, MA, USA: MIT Press, 1996. (Cited on pages 16, 26, 29, 32, 33, 43, 189, and 190.)

[138] R. Pfeifer and J. Bongard. *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, MA, USA: MIT Press, 2007. (Cited on pages 2, 16, 26, 151, 154, and 189.)

[139] R. Pfeifer, M. Lungarella, and F. Iida. Self-organization, embodiment, and biologically inspired robotics. *Science*, 318(5853):1088–1093, 2007. (Cited on pages 93, 189, and 190.)

[140] R. Pfeifer and C. Scheier. *Understanding Intelligence*. Cambridge, MA, USA: MIT Press, 1999. (Cited on pages 16, 33, and 189.)

[141] J. Piater, M. Steedman, and F. Wörgötter. Learning in PACO-PLUS. Technical report, PACO-PLUS Technical Report, 2009, [Accessed January 31th, 2019]. Available at URL `http://www.paco-plus.org` under “Object-Action-Complexes”. (Cited on page 196.)

[142] L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Stockholm, Sweden, pages 3406–3413, 2016. (Cited on page 27.)

[143] C. G. Prince, N. A. Helder, and G. J. Hollich. Ongoing emergence: A core concept in epigenetic robotics. In L. Berthouze, F. Kaplan, H. Kozima, H. Yano, J. Konczak, G. Metta, J. Nadel, G. Sandini, G. Stojanov, and C. Balkenius, editors, *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, volume 123, pages 63–70. Lund, Sweden: Lund University Cognitive Studies, 2005. (Cited on pages 4, 5, 26, 30, 35, and 40.)

[144] K. Ramirez-Amaro, M. Beetz, and G. Cheng. Automatic segmentation and recognition of human activities from observation based on semantic reasoning. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Chicago, IL, USA, pages 5043–5048, 2014. (Cited on page 188.)

[145] G. Rizzolatti, G. Luppino, and M. Matelli. The organization of the cortical motor system: New concepts. *Electroencephalography and Clinical Neurophysiology*, 106(4):283–296, 1998. (Cited on page 120.)

[146] P. Rocca, G. Oliveri, and A. Massa. Differential evolution as applied to electromagnetics. *IEEE Antennas and Propagation Magazine*, 53(1):38–49, 2011. (Cited on page 52.)

[147] M. Rolf and J. J. Steil. Efficient exploratory learning of inverse kinematics on a bionic elephant trunk. *IEEE Transactions on Neural Networks and Learning Systems*, 25(6):1147–1160, 2014. (Cited on pages 45 and 102.)

[148] M. Rolf, J. J. Steil, and M. Gienger. Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, 2(3):216–229, 2010. (Cited on pages 19, 45, and 102.)

[149] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. In T. A. Polk and C. M. Seifert, editors, *Cognitive Modeling*, pages 213–220. Cambridge, MA, USA: MIT Press, 2002. (Cited on pages 25, 57, and 159.)

[150] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Cambridge, MA, USA: MIT Press, 1986. (Cited on pages 25 and 57.)

[151] R. Saegusa, G. Metta, and G. Sandini. Own body perception based on visuomotor correlation. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, pages 1044–1051, 2010. (Cited on pages 19, 21, and 45.)

[152] R. Saegusa, G. Metta, G. Sandini, and L. Natale. Developmental perception of the self and action. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):183–202, 2014. (Cited on pages 19 and 21.)

[153] R. Saegusa, G. Metta, G. Sandini, and S. Sakka. Active motor babbling for sensorimotor learning. In *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, Bangkok, Thailand, pages 794–799, 2009. (Cited on pages 43 and 45.)

[154] G. Sandini, G. Metta, and D. Vernon. RobotCub: An open framework for research in embodied cognition. In *Proceedings of the IEEE International Conference on Humanoid Robots*, Santa Monica, CA, USA, volume 1, pages 13–32, 2004. (Cited on page 29.)

[155] K. Sasaki, H. Tjandra, K. Noda, K. Takahashi, and T. Ogata. Neural network based model for visual-motor integration learning of robot's drawing behavior: Association of a drawing motion from a drawn image. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Hamburg, Germany, pages 2736–2741, 2015. (Cited on page 48.)

[156] J. Saunders, H. Lehmann, F. Förster, and C. L. Nehaniv. Robot acquisition of lexical meaning — moving towards the two-word stage. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, San Diego, CA, USA, pages 1–7, 2012. (Cited on page 197.)

[157] B. Scassellati. Theory of mind for a humanoid robot. *Autonomous Robots*, 12(1):13–24, 2002. (Cited on page 195.)

[158] G. Schillaci. *Sensorimotor Learning and Simulation of Experience as a Basis for the Development of Cognition in Robotics.* Ph.D. thesis, Mathematisch-Naturwissenschaftliche Fakultät II, Humboldt-Universität zu Berlin, Berlin, Germany, 2014. (Cited on pages 102 and 151.)

[159] M. Schmerling, G. Schillaci, and V. V. Hafner. Goal-directed learning of hand-eye coordination in a humanoid robot. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Providence, RI, USA, pages 168–175, 2015. (Cited on pages 8, 19, 92, and 102.)

[160] J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006. (Cited on page 185.)

[161] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. (Cited on page 93.)

[162] G. Schöner. Development as change of system dynamics: Stability, instability, and emergence. In J. P. Spencer, M. S. C. Thomas, and J. L. McClelland, editors, *Toward a Unified Theory of Development: Connectionism and Dynamic Systems Theory Re-Considered*, pages 25–48. Oxford Series in Developmental Cognitive Neuroscience. New York, NY, USA: Oxford University Press, 2009. (Cited on page 190.)

[163] S. H. Scott. Optimal feedback control and the neural basis of volitional motor control. *Nature Reviews Neuroscience*, 5(7):534–546, 2004. (Cited on page 23.)

[164] M. Shanahan. A cognitive architecture that combines internal simulation with a global workspace. *Consciousness and Cognition*, 15(2):433–449, 2006. (Cited on pages 39, 40, 41, 42, and 194.)

[165] M. Shanahan and B. Baars. Applying global workspace theory to the frame problem. *Cognition*, 98(2):157–176, 2005. (Cited on page 39.)

[166] S. C. Shapiro and J. P. Bona. The GLAIR cognitive architecture. *International Journal of Machine Consciousness*, 2(2):307–332, 2010. (Cited on page 193.)

[167] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, Anchorage, AK, USA, pages 69–73, 1998. (Cited on page 52.)

[168] Y. Q. Shi and H. Sun. *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards*. Boca Raton, Florida, USA: CRC Press, 1999. (Cited on page 22.)

[169] K. Shima, M. Isoda, H. Mushiake, and J. Tanji. Categorization of behavioural sequences in the prefrontal cortex. *Nature*, 445(7125):315–318, 2007. (Cited on page 23.)

[170] S. Shipp, R. A. Adams, and K. J. Friston. Reflections on agranular architecture: Predictive coding in the motor cortex. *Trends in Neurosciences*, 36(12):706–716, 2013. (Cited on page 154.)

[171] P. K. Simpson. *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*. Oxford, UK: Pergamon Press, 1990. (Cited on page 190.)

[172] A. W. Smitsman and D. Corbetta. Action in infancy — perspectives, concepts, and challenges. In J. G. Bremner and T. D. Wachs, editors, *The Wiley-Blackwell Handbook of Infant Development*, volume 1, 2nd edition, pages 167–203. Hoboken, NJ, USA: Blackwell Publishing Ltd., 2010. (Cited on pages 2, 151, and 152.)

[173] Softbank Robotics. NAO — versions and body type, [Accessed January 29th, 2019]. Available at URL `http://doc.aldebaran.com/2-1/family/body_type.html`. (Cited on pages 91 and 151.)

[174] SoftBank Robotics. NAO joints — Aldebaran 2.1.4.13 documentation, [Accessed January 29th, 2019]. Available at URL `http://doc.aldebaran.com/2-1/family/robots/joints_robot.html`. (Cited on page 106.)

[175] SoftBank Robotics. NAO the humanoid robot | SoftBank Robotics EMEA, [Accessed January 29th, 2019]. Available at URL `https://www.softbankrobotics.com/emea/en/nao`. (Cited on pages 8, 42, and 167.)

[176] F. J. Solis and R. J.-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981. (Cited on page 52.)

[177] R. Storn. On the usage of differential evolution for function optimization. In *Proceedings of the Biennial Conference of the North American Fuzzy Information Processing Society*, pages 519–523. IEEE, 1996. (Cited on page 52.)

[178] R. Storn and K. Price. Differential evolution — a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, TR-95-012, ICSI, International Computer Science Institute, Berkeley, CA, USA, 1995. (Cited on page 52.)

[179] R. Storn and K. Price. Differential evolution — a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997. (Cited on pages 52 and 62.)

[180] A. Stoytchev. Some basic principles of developmental robotics. *IEEE Transactions on Autonomous Mental Development*, 1(2):122–130, 2009. (Cited on pages 5, 27, 40, and 154.)

[181] H. Sumioka, Y. Yoshikawa, and M. Asada. Reproducing interaction contingency toward open-ended development of social actions: Case study on joint attention. *IEEE Transactions on Autonomous Mental Development*, 2(1):40–50, 2010. (Cited on page 190.)

[182] R. Sun. *Duality of the Mind: A Bottom-Up Approach Toward Cognition*. Mahwah, NJ, USA: Lawrence Erlbaum Associates, 2002. (Cited on page 30.)

[183] R. Sun. Desiderata for cognitive architectures. *Philosophical Psychology*, 17(3):341–373, 2004. (Cited on page 30.)

[184] R. Sun. The importance of cognitive architectures: An analysis based on CLARION. *Journal of Experimental and Theoretical Artificial Intelligence*, 19(2):159–193, 2007. (Cited on page 195.)

[185] R. S. Sutton. Verification (November 14th, 2001), [Accessed January 29th, 2019]. Available at URL `http://incompleteideas.net/IncIdeas/Verification.html`. (Cited on pages 27 and 154.)

[186] R. S. Sutton. Verification, the key to AI (November 15th, 2001), [Accessed January 29th, 2019]. Available at URL `http://incompleteideas.net/IncIdeas/KeytoAI.html`. (Cited on pages 27 and 154.)

[187] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998. (Cited on page 18.)

[188] K. Takahashi, T. Ogata, H. Tjandra, Y. Yamaguchi, and S. Sugano. Tool-body assimilation model based on body babbling and neurodynamical system. *Mathematical Problems in Engineering*, 2015(837540):1–15, 2015. (Cited on pages 48 and 102.)

[189] K. Takahashi, T. Ogata, H. Yamada, H. Tjandra, and S. Sugano. Effective motion learning for a flexible-joint robot using motor babbling. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Hamburg, Germany, pages 2723–2728, 2015. (Cited on page 48.)

[190] J. Tani. Model-based learning for mobile robot navigation from the dynamical systems perspective. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 26(3):421–436, 1996. (Cited on pages 40, 41, 42, and 45.)

[191] J. Tani. Learning to generate articulated behavior through the bottom-up and the top-down interaction processes. *Neural Networks*, 16(1):11–23, 2003. (Cited on pages 24, 40, 41, 42, and 45.)

[192] J. Tani. *Exploring Robotic Minds: Actions, Symbols, and Consciousness as Self-Organizing Dynamic Phenomena*. Oxford Series on Cognitive Models and Architectures. New York, NY, USA: Oxford University Press, 2016. (Cited on pages 4, 5, 7, 24, 25, 32, 44, 48, 91, 92, 93, 154, and 181.)

[193] J. Tani. Exploring robotic minds by predictive coding principle. In P.-Y. Oudeyer, editor, *IEEE CDS Newsletter — The Newsletter of the Technical Committee on Cognitive and Developmental Systems*, volume 14, number 1, pages 4–5, 2017. (Cited on pages 4, 7, 91, 93, and 181.)

[194] J. Tani, M. Ito, and Y. Sugita. Self-organization of distributedly represented multiple behavior schemata in a mirror system: Reviews of robot experiments using RNNPB. *Neural Networks*, 17(8–9):1273–1289, 2004. (Cited on page 24.)

[195] J. Tani, R. Nishimoto, J. Namikawa, and M. Ito. Codevelopmental learning between human and humanoid robot using a dynamic neural-network model. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, 38(1):43–59, 2008. (Cited on pages 36, 37, and 44.)

[196] J. Tani, R. Nishimoto, and R. W. Paine. Achieving “organic compositionality” through self-organization: Reviews on brain-inspired robotics experiments. *Neural Networks*, 21(4):584–603, 2008. (Cited on pages 23 and 48.)

[197] E. Thelen and L. B. Smith. *A Dynamic Systems Approach to the Development of Cognition and Action*. Cambridge, MA, USA: MIT Press, 1996. (Cited on pages 32, 93, and 190.)

[198] M. S. C. Thomas and M. H. Johnson. New advances in understanding sensitive periods in brain development. *Current Directions in Psychological Science*, 17(1):1–5, 2008. (Cited on page 2.)

[199] M. Toda. *Man, Robot, and Society: Models and Speculations*. Leiden, Netherlands: Martinus Nijhoff Publishers, 1982. (Cited on pages 26, 29, and 43.)

[200] A. Törn and A. Žilinskas. *Global Optimization*. Part of the Lecture Notes in Computer Science book series, volume 350. Berlin, Germany: Springer, 1989. (Cited on page 75.)

[201] I. C. Trelea. The particle swarm optimization algorithm: Convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, 2003. (Cited on page 52.)

[202] E. Ugur, Y. Nagai, E. Sahin, and E. Oztop. Staged development of robot skills: Behavior formation, affordance learning and imitation with motionese. *IEEE Transactions on Autonomous Mental Development*, 7(2):119–139, 2015. (Cited on pages 26, 40, 41, 42, 44, and 45.)

[203] E. Ugur, E. Oztop, and E. Sahin. Goal emulation and planning in perceptual space using learned affordances. *Robotics and Autonomous Systems*, 59(7–8):580–595, 2011. (Cited on pages 19, 20, and 21.)

[204] E. Ugur, E. Sahin, and E. Oztop. Self-discovery of motor primitives and learning grasp affordances. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, pages 3260–3267, 2012. (Cited on pages 20 and 21.)

[205] F. J. Varela, E. Thompson, and E. Rosch. *The Embodied Mind: Cognitive Science and Human Experience.* Cambridge, MA, USA: MIT Press, 1991. (Cited on page 43.)

[206] D. Vernon. Enaction as a conceptual framework for developmental cognitive robotics. *Paladyn Journal of Behavioral Robotics*, 1(2):89–98, 2010. (Cited on pages 8, 29, 35, 38, 45, 92, 189, and 194.)

[207] D. Vernon. Reconciling autonomy with utility: A roadmap and architecture for cognitive development. In A. V. Samsonovich and K. R. Jóhannsdóttir, editors, *Proceedings of the International Conference on Biologically Inspired Cognitive Architectures*, volume 233 of *Frontiers in Artificial Intelligence and Applications*, pages 412–418. Amsterdam, Netherlands: IOS Press, 2011. (Cited on pages 8, 29, 35, 36, 38, 45, 92, and 194.)

[208] D. Vernon. *Artificial Cognitive Systems: A Primer*. Cambridge, MA, USA: MIT Press, 2014. (Cited on pages xxii, 30, 31, 155, 187, 189, and 191.)

[209] D. Vernon, G. Metta, and G. Sandini. The iCub cognitive architecture: Interactive development in a humanoid robot. In *Proceedings of the Sixth IEEE International Conference on Development and Learning*, London, UK, pages 122–127, 2007. (Cited on pages 8, 35, 38, 45, 92, and 194.)

[210] D. Vernon, G. Metta, and G. Sandini. A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. *IEEE Transactions on Evolutionary Computation*, 11(2):151–180, 2007. (Cited on pages xxii, 28, 29, 30, 31, 33, 34, 50, 155, 187, 189, and 191.)

[211] D. Vernon, C. von Hofsten, and L. Fadiga. *A Roadmap for Cognitive Development in Humanoid Robots*, volume 11 of Cognitive Systems Monographs. R. Dillmann, Y. Nakamura, S. Schaal, and D. Vernon, editors. Berlin, Germany: Springer, 2010. (Cited on pages 1, 8, 31, 33, 35, 36, 38, 43, 44, 45, 92, 93, 102, 151, 187, 189, and 194.)

[212] J. Vesterstrom and R. Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, Portland, OR, USA, volume 2, pages 1980–1987, 2004. (Cited on page 52.)

[213] Y. Wang, X. Wu, and J. Weng. Skull-closed autonomous development. In B.-L. Lu, L. Zhang, and J. Kwok, editors, *Neural Information Processing (ICONIP)*, volume 7062 of *Lecture Notes in Computer Science*, pages 209–216. Berlin, Germany: Springer, 2011. (Cited on page 50.)

[214] J. Weng. A theory for mentally developing robots. In *Proceedings of the Second International Conference on Development and Learning*, Cambridge, MA, USA, pages 131–140, 2002. (Cited on page 194.)

[215] J. Weng. Developmental robotics: Theory and experiments. *International Journal of Humanoid Robotics*, 1(2):199–236, 2004. (Cited on pages 16, 17, 26, and 27.)

[216] J. Weng. Symbolic models and emergent models: A review. *IEEE Transactions on Autonomous Mental Development*, 4(1):29–54, 2012. (Cited on pages 32 and 50.)

[217] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Autonomous mental development by robots and animals. *Science*, 291(5504):599–600, 2001. (Cited on pages 1, 4, 16, 26, and 194.)

[218] J. Weng and S. Zeng. A theory of developmental mental architecture and the DAV architecture design. *International Journal of Humanoid Robotics*, 2(2):145–179, 2005. (Cited on pages 190 and 194.)

[219] E. Wieser and G. Cheng. Forming goal-directed memory for cognitive development. In *Proceedings of Humanoids 2012 Workshop on Developmental Robotics: Can developmental robotics yield human-like cognitive abilities?*, pages 38–39. Workshop at the *IEEE International Conference on Humanoid Robots*, Osaka, Japan, 2012. (Cited on pages 96, 151, 164, and 178.)

[220] E. Wieser and G. Cheng. Predictive action selector for generating meaningful robot behaviour from minimum amount of samples. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Genoa, Italy, pages 139–145, 2014. (Cited on pages 92, 123, 129, 152, and 158.)

[221] E. Wieser and G. Cheng. Progressive learning of sensory-motor maps through spatiotemporal predictors. In *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, Cergy-Pontoise, Paris, France, pages 43–48, 2016. (Cited on pages 92, 129, 158, 161, and 162.)

[222] E. Wieser and G. Cheng. A self-verifying cognitive architecture for robust bootstrapping of sensory-motor skills via multipurpose predictors. *IEEE Transactions on Cognitive and Developmental Systems*, 10(4):1081–1095, 2018. (Cited on pages 39, 40, 41, 42, 92, 151, and 166.)

[223] E. Wieser and G. Cheng. EO-MTRNN: Evolutionary optimization of hyperparameters for a neuro-inspired computational model of spatiotemporal learning. *Biological Cybernetics*, 2020. `https://doi.org/10.1007/s00422-020-00828-8`. (Cited on page 47.)

[224] G. Wiest, M.-A. Amorim, D. Mayer, S. Schick, L. Deecke, and W. Lang. Cortical responses to object-motion and visually-induced self-motion perception. *Cognitive Brain Research*, 12(1):167–170, 2001. (Cited on page 116.)

[225] S. P. Wise and E. A. Murray. Arbitrary associations between antecedents and actions. *Trends in Neurosciences*, 23(6):271–276, 2000. (Cited on page 120.)

[226] D. M. Wolpert, Z. Ghahramani, and M. I. Jordan. An internal model for sensorimotor integration. *Science*, 269(5232):1880–1882, 1995. (Cited on pages 22 and 102.)

[227] D. M. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7–8):1317–1329, 1998. (Cited on pages 18, 23, and 39.)

[228] F. Wörgötter, A. Agostini, N. Krüger, N. Shylo, and B. Porr. Cognitive agents — a procedural perspective relying on the predictability of object-action-complexes (OACs). *Robotics and Autonomous Systems*, 57(4):420–432, 2009. (Cited on pages 19 and 196.)

[229] F. Wörgötter, C. Geib, M. Tamosiunaite, E. E. Aksoy, J. Piater, H. Xiong, A. Ude, B. Nemec, D. Kraft, N. Krüger, M. Wächter, and T. Asfour. Structural bootstrapping — a novel, generative mechanism for faster and more efficient acquisition of action-knowledge. *IEEE Transactions on Autonomous Mental Development*, 7(2):140–154, 2015. (Cited on pages 26 and 27.)

[230] Y. Yamashita and J. Tani. Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Computational Biology*, 4(11):e1000220, 2008. (Cited on pages xvii, 6, 9, 25, 36, 37, 38, 40, 41, 42, 43, 44, 45, 48, 49, 50, 53, 57, 83, 84, 85, 86, 87, 88, 89, 182, 183, and 190.)

[231] Y. Yamashita and J. Tani. Spontaneous prediction error generation in schizophrenia. *PLoS One*, 7(5):e37843, 2012. (Cited on page 48.)

[232] X. Yao, Y. Liu, and G. Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999. (Cited on pages 52 and 75.)

[233] X. Yao and Y. Xu. Recent advances in evolutionary computation. *Journal of Computer Science and Technology*, 21(1):1–18, 2006. (Cited on page 52.)

[234] R. S. Zemel and T. J. Sejnowski. A model for encoding multiple object motions and self-motion in area MST of primate visual cortex. *The Journal of Neuroscience*, 18(1):531–547, 1998. (Cited on pages 116 and 122.)

[235] T. Ziemke and R. Lowe. On the role of emotion in embodied cognitive architectures: From organisms to robots. *Cognitive Computation*, 1(1):104–117, 2009. (Cited on page 194.)