Rikke Amilde Løvlid

Internal Models as Echo State Networks

Learning to Execute Arm Movements

Thesis for the degree of Philosophiae Doctor

Trondheim, Desember 2013

Norwegian University of Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Computer and Information Science

NTNU – Trondheim
Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 978-82-471-4810-5 (printed ver.)
ISBN 978-82-471-4811-2 (electronic ver.)
ISSN 1503-8181

Doctoral theses at NTNU, 2013:335

Printed by NTNU-trykk

ABSTRACT

As robots are becoming more and more complex, with higher degrees-of-freedom, lighter limbs, and springy joints, it becomes harder to control their movements. New approaches, inspired from neuroscience, are attracting increased attention among computer scientists dealing with motor control.

The focus in this thesis is on how robots can learn to control their limbs by learning how their bodies work, i.e., by learning internal models of their motor apparatus. Inspiration from cerebellar research combined with concepts from traditional control theory has been used as a basis.

The research in the thesis is twofold. First, we investigate how internal models can be used to solve different control problems. In particular, we consider how to handle delays in the sensory-motor-loop and how to realize bimanual coordination. Second, we study how the internal models can be represented and learned. This includes how to choose movements to learn from in order to learn as much of the internal model as possible and how to actually learn the training movement.

A simple simulator is used in the experiments and the simulator's internal models were implemented as echo state networks (ESNs), a relatively new and promising type of recurrent neural networks. The simulator learns internal modes of his motor apparatus by imitating human motion. Human motion data was recorded and the task of the simulator's control system is to generate motor commands that result in the simulator replicating the recorded movement.

From the experiments we conclude that using ESNs for representing and learning internal models looks promising. With an ESN we are able to generalize to imitating novel movements, and we demonstrate that it is able to learn various bimanual coordination patterns. However, training ESNs is challenging and a major contribution from this thesis is a novel training method that works particularly well in our application. The thesis also contributes to how different internal models can be used and trained together.

PREFACE

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfillment of the requirements for the degree of *philosophiae doctor*.

This doctoral work has been performed at the Department of Computer and Information Science, NTNU, Trondheim, with Associate Professor Pinar Özturk as main supervisor and with co-supervisors Professor Agnar Aamodt and Professor Pauline Haddow.

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Pinar Öztürk. I appreciate her honest, straightforward feedback and constructive criticisms. Also, she was always available, answered emails on Sundays, and still kept the spirit up at 3 o'clock in the morning before a deadline. I also would like to thank my co-supervisor Pauline Haddow, who provided valuable insights on what is expected of a PhD thesis.

At IDI I had many inspiring co-PhD-students. Axel Tidemann, Lester Solbakken and Boye Annfelt Høverstad are some of the hard working people I had to look up to.

My colleagues at Simula School of Research and innovation gave me extra inspiration to get through, and my colleagues at FFI made it impossible not to finish. Especially my boss Karsten Bråthen has been very encouraging and mildly pushy.

I also want to thank Ingeborg Skaret Kjos-Hanssen, who has educated me in scientific writing, Lene Bertheussen and Guro Kristin Svendsen for giving feedback on the final thesis, and Solveig Bruvoll for proofreading.

Last, but not least, I want to thank my boyfriend Rolv Seehuus, who has also been proofreading as well as listening to my endless train of thoughts and anxiety, looking after our son Emrik so that I could work, and never stopped encouraging me to go on. I love you.

CONTENTS

I

Research Introduction and Overview

INTRODUCTION 1

This thesis is about robots learning to control their limbs by learning how their bodies work. It is about how inspiration from neuroscience can be used for robot motor control and how control structures can be represented and learned. The main part of the thesis is a collection of papers included in part II. The motivation behind the research, the overall research questions and background theory is included in part I together with a summary of the papers and discussion and conclusion of conducted research. In this first chapter the addressed challenges are introduced and research questions are motivated.

1.1 Introduction

Robots are becoming more and more complex every day, with higher degrees-of-freedom bodies, lighter limbs and springy joints. These new types of robots have the potential to evolve from factory floors into our everyday lives. However, their complex bodies make it hard to deliver accurate analytical models of their motor apparatus to ensure stable control of their limbs [75]. New control schemes are required, and as our brains solve these control challenges perfectly, neuroscience is a natural place for inspiration.

The control architectures used in this thesis do not aim to be biologically accurate, but they are biologically inspired. In order to set our research in the perspective of human motor control, I would like to start with a short, simplified introduction to motor control in the human brain. A comprehensive discussion of this topic can be found in a recent textbook on computational motor control [66].

Shadmehr and Krakauer suggest that in order to execute a movement, our brain needs to solve three kinds of problems: 1) It needs to be able to accurately predict the sensory consequences of the motor commands. 2) It needs to combine those predictions with actual sensory feedback to form a belief about the state of the body and the world, and 3) given this belief about the state of the body and the world, it has to adjust the gains of senso-

rimotor feedback loops so that the movement maximizes some measure of performance [67]. These problems are solved by cooperation of the motor cortex, the cerebellum and the basal ganglia. In addition, lower level motor control takes place in the brain stem and the spinal cord.

The functions of the cerebral cortex, the basal ganglia and the cerebellum are not limited to motor control but also involve other cognitive tasks. Anatomical studies have revealed discrete circuits or loops that reciprocally interconnect a large and diverse set of cerebral cortical areas with the basal ganglia and cerebellum. The individual loops appear to be involved in distinct behavioral functions [51]. From a computational viewpoint, it has been suggested that these different parts of the brain are specialized for different types of learning: unsupervised learning in the cerebral cortex, reward based, reinforcement learning in the basal ganglia and error-based, supervised learning in the cerebellum [14, 15, 16]. How their contributions to motor control are reflected in this division will be explained next.

The motor cortex is the primary part of cerebral cortex involved in motor control. It can be described as a set of feedback controllers [67], which directly calculates motor commands by comparing the desired position of limbs with their estimated position. The motor cortex works as the master of the motor control system, and it has the necessary means for calculating the motor commands and getting the general shape of the trajectory itself, but it uses the basal ganglia and the cerebellum as consultants with different specialties. The motor commands originating from the cerebral cortex are optimized in terms of their reward value and sensorimotor accuracy by going through the basal ganglia and cerebellar loop circuits, respectively.

The basal ganglia learns rewards associated with states and actions and contributes by selecting appropriate actions and suppressing unnecessary actions by predicting the reward values. It is associated with sequence learning, the acquirement of habits, and chunking of actions [22, 23, 24, 64]. Prescott et al. use a model of the basal ganglia for robot control [59], where the task of the control system is to generate coherent sequences of actions based on input salience values for each of five possible actions. The salience values or urgency signals of the different possible actions are calculated as a weighted sum of relevant perceptual and motivational signals as well as a signal indicating that the system is in the middle of an action consisting of multiple steps. Prescott et al. did not use reinforcement learning, or any other kind of learning, but tuned the parameters by hand in order to achieve “biologically plausible” behavior.

Where the basal ganglia decides “what” to do, the cerebellum is involved in calculating “how” to realize the selected action. To do this it learns internal models. An internal model is a model that mimics a natural process and can be acquired by supervised learning. In motor control theory one talks about two types of internal models, *forward models* and *inverse models*. Forward models predict the consequences of motor commands in terms of positions, whereas inverse models generate the motor commands that will result

in the desired positions. Whether the internal modes in cerebellum are forward models [50], inverse models [37, 34] or both [82, 28, 76] is still an open question. By using the internal models, cerebellum is believed to transform a well-articulated plan into graceful coordinated movement [1, 5]. Smagt gives an overview of different cerebellar models intended for robot control and discusses when these models might be useful [74].

1.2 Motivation

On the route to improved robotic motor control, this thesis contributes to investigations into the use of internal models. The research can be divided into two aspects, 1) how to use internal models to solve various control problems and 2) how to represent and learn the internal models.

We chose to look into two particular control problems and how internal models can be used to solve them, namely *delays in the sensory-motor-loop* and *bimanual coordination*. These are motivated in sections 1.2.1 and 1.2.2 respectively. The problem with delays in the sensory-motor-loop is about how to handle the fact that consequences of one's own motor commands might be perceived with a delay that could affect proper timing of new motor commands. Bimanual coordination is another control problem that deals with how to generate proper coordinated movements with two arms.

In order to be able to use internal models for complex robots, the internal models must be learned. We represent the internal models as *echo state networks* (ESNs), a relatively new and promising type of recurrent neural networks, which will be explained in section 2.2. Our first focus was how to choose training movements in order to maximize the network's generalization capability. Motivation and related work for this problem is presented in section 1.2.3. Making the ESN learn the training movement proved to be challenging. This problem is further explained in section 1.2.4.

1.2.1 Handling Delays in the Sensory-Motor Loop

The sensory-motor system in humans is able to adjust for the presence of noise and delay in sensory feedback, as well as for changes in the body and the environment that alter the relationship between motor commands and their sensory consequences. This adjustment is achieved by employing anticipatory mechanisms based on the concept of internal models. In robotics the ability to predict the consequences of actions has been used to *"mentally" simulate alternative sensory-motor sequences* and to *handle delays in the sensory-motor loop*. In our experiments we only used prediction to handle delays in the sensory-motor loop, but we will also give an example of using prediction to choose between different actions, as the solutions are very similar.

An example of how a robot can use mental simulation for action selection is described in a paper by Hoffman [27]. Here, a robot learns to predict how its visual input changes during movement. By generating "mental" images based on a sequence of motor commands, the robot is able to calculate the distance to an obstacle or recognizing a dead end. The prediction is made by a forward model implemented as a feed-forward neural network. Similarly, Gross et al. provided a neural control architecture that learns to predict and evaluate the sensory consequences of hypothetically executed actions by simulating alternative sensory-motor sequences, selecting the best one, and executing it in reality [26].

When using prediction to handle delays in the sensory-motor-loop, the task is not to predict the consequences of hypothetical actions, but to predict the consequences of the actual executed action. This prediction is needed when it takes time before the robot is able to perceive these consequences, and it might need to issue a new action before it perceives the result of the previous. Generally, the sensory motor loop in robotics can be divided into four steps, *sensor acquisition*, *sensor processing*, *motor command generator* and *actuation*. The overall time of the sensory-motor loop is the sum of the time these four steps take. Especially in vision data processing, the time needed for sensor-processing can be relatively high, which affects the robot's capability to react in real time [8]. To avoid adverse consequences of this delay, Datteri et al. propose a reactive control scheme that includes a forward model, or *expected perception generator*, as it is called in their papers [8, 9, 7]. The forward model predicts the sensory input, a visual image. This prediction is compared to the actual visual image before it is further processed. If the predicted image matches the actual image, the sensor-processing step is skipped, as the situation is as expected. If the prediction does not match, the movement is stopped. No match means that some unexpected event is affecting the current task execution, and the traditional sensory data processing must be carried out in order to find out how to compensate and get the movement back on track. The same approach of trying to detect when something unexpected happens was also described in [41]. In the experiments a model of human sensory-motor coordination in grasping was implemented. The robot learned to reach and grasp an object detected by vision and to predict the tactile feedback by means of a forward model implemented as a neuro-fuzzy network. The intention was to use the predicted tactile feedback in a similar control system as the one proposed by Datteri et al. [8]. If there is a mismatch between the predicted and actual tactile feedback, compensatory actions should be triggered and internal models of the observed object and the hand should be updated.

The expected perception strategy uses prediction to decide whether the time consuming sensory-processing step can be skipped. It is assumed that the robot can be controlled without sensory feedback as long as nothing unexpected (like external forces) happens. Contrary to this research, we will in this thesis assume that the inverse model needs sensory feedback also when nothing unexpected happens. Our suggestion is to predict the result of the sensory processing step, and use this predicted state together with the delayed, actual result of the sensory processing step as sensory feedback to the inverse model. A similar

solution was proposed by Wolpert et. al [80]. They suggested using two forward models. The first predicts the next state, i.e., the result of the sensory processing step, based on the estimated current state and the motor command. The second predicts the sensory input from the estimated current state. The sensory prediction error is translated into a state error with a Kalman filter, and this state error is used to alter the predicted state. We did not compare this approach to ours as in our experiments the sensory feedback and the state representation were the same, meaning contrary to for example using an image as sensory feedback, the sensory feedback consisted of a vector expressing the position of the arms, and the state was represented with the same type of vector. The sensory processing was just simulated by artificially delaying the sensory feedback.

In our solution we propose using both the fast, predicted feedback and the slow, actual feedback as input to the inverse model. The idea is that the delayed feedback can compensate for errors in the forward model, and that the predicted state together with the delayed sensory feedback make a better estimate of the current state than any of them would alone. This is in contrast to the classical Smith Predictor control scheme [50], where only the predicted feedback is used for control, whereas the actual feedback is only used to update the forward model. The Smith Predictor control scheme on the other hand, addresses another problem, namely how to estimate the delay in order to calculate the error for training the forward model. The Smith Predictor control scheme consists of a controller and two forward models. The first forward model predicts the next state (e.g., arm position) based on an efferent copy of the motor command together with the current state. This prediction is used as input to the controller. The second forward model models the delay in the sensory-motor loop. The prediction from the first model is used as input to the second model. The second model delays the predicted state so that it can be compared with the actual state, which is based on sensory feedback. The difference between the predicted and the actual state is used to update the internal models. In our experiments we assumed the duration of the delay was known, and that we therefore did not need the second forward model predicting the delay. What is most accurate in practice, measuring the delay or learning to predict it, is, however, an open questions that is out of scope of this thesis.

1.2.2 Bimanual Coordination

The second control problem addressed in this thesis is bimanual coordination, and our objective was to study how to apply internal models to solve this problem. In this section I will discuss the traditional approach to bimanual control and present two studies that address problems that appear as the robots are getting more complex before relating this to our research.

Most research on bimanual coordination regards the problem of using two arms to manipulate some object, where the task is to calculate the movement trajectory for the two arms in order to move the object, or maybe assemble two objects. From the viewpoint of con-

ventional, analytical robot trajectory planning, two-arms manipulation is a relatively well research topic. Suggested control schemes can be divided into two groups, “master-lave schemes” and “symmetrical solutions”, which are compared in papers [10] and [40]. In a master-slave scheme the master arm is typically moved along a desired trajectory, and the slave arm follows the movement of the master arm by maintaining a force relative to the manipulated object. In symmetric control schemes both arms are controlled based on both their relative position and the force directed at the object they are manipulating. However, these traditional, analytical control schemes were designed to deal with limited DOFs.

Morasso and Mohan et al. suggest tackling the problem of increased number of DOFs by applying the “passive motion” paradigm to bimanual coordination [54, 53]. The passive motion paradigm is an alternative to optimal control theory, where a movement trajectory is chosen among several possibilities by minimizing some cost function [21, 73]. The passive motion paradigm suggests coordinating the DOFs by inducing a virtual force field applied to a small number of relevant parts of the body. The internal model creating the trajectory operates on this small set of force fields, instead of all DOFs, in analogy to controlling a marionette by means of attached strings [52]. Morasso and Mohan et al. apply the passive motion paradigm to bimanual coordination by using mutual force fields for both arms, e.g., by connecting a force field to the object that is manipulated by both arms.

None of the above studies address learning bimanual movement skills. Gribovskaya and Billard, on the other hand, suggest learning coordinated movements by extracting spatial and temporal constraints from observed movement patterns and use these constraints when generating the movement trajectories [25]. Contrary to this approach, we predefined the constraints relating the movement of the two arms and investigated what kind of constraints the neural network was able to learn. As in the above examples, the controller of one arm had access to the position of the opposite arm.

1.2.3 Choosing Movements to Learn From

Both when working on how to handle delays in the sensory-motor-loop and how to realize bimanual coordination we used internal models. A common challenge addressed in this thesis is how to learn these internal models. In particular we concentrated on learning the inverse model, because learning an inverse model is generally harder than learning a forward model. As mentioned, a forward model predicts the consequences of actions, whereas an inverse model calculates the action that will lead to some desired consequence. Learning the inverse model is harder, because there might be multiple possible actions that lead to the same result, making the inverse model ill defined.

A robot might learn the inverse model of its motor apparatus by issuing random motor commands and observe what happens [11]. How the brain does it is still an open question.

Although this *motor babbling* is one hypothesis in developmental psychology [49], other findings suggest that children use more goal directed movements [79]. Several different strategies are used in robotics and will be explained next.

D'Souza, Vijayakumar and Schaal used locally weighted projection regression (LWPR) [78] to learn the inverse kinematics model of a humanoid robot [17]. The inverse model was learned online by initially biasing the motion toward a default posture. This default posture was also used to bias the solution to the inverse kinematics problem toward a "natural posture". The system was first trained on data collected from motor babbling. The system was then tested on the "figure-eight" generation task, i.e., generation of the shape of the number 8, which is a recurring exercise for recurrent neural networks (RNNs) [57]. Performance was not perfect, because the joint space was large and the motor babbling only covered sparse data from the region required by the "figure-eight" task. Not surprisingly, better results were obtained by training on the "figure-eight" task itself.

If the robot is only going to do a limited number of movements as in the above example, they do not have to learn the whole inverse model, and they will do better only training on the movements they will perform. Exploration can then be guided by humans demonstrating the movements [6] or a programmer specifying a reward function [58]. One method for learning only the part of the inverse model relevant for given trajectories is the shifting setpoint algorithm [62]. It uses motor babbling to build the inverse model along tubes in actuator space, from start positions to goal positions.

However, the robot might need to learn a variety of movements over a long period of time. Active learning algorithms might be beneficial when it is hard to predict what kind of training data the robot will need [65]. These algorithms generate or select training data themselves, without humans having to specify goals or training trajectories.

Also, when the robot needs to learn a large part of the inverse model, it becomes important to minimize the number of training examples required to reach a certain level of performance. The Self-Adaptive Goal Generation - Robust Intelligent Adaptive Curiosity (SAGG-RIAC) is an active learning algorithm that uses motor babbling in the task space (e.g., arm positions in Cartesian coordinates) as opposed to motor babbling in the actuator space (e.g., joint angles) [4]. Baranes and Oudeyer showed that exploration in the task space can be a lot faster than exploration in actuator space for learning the inverse kinematics of a redundant robot [4].

The general assumption in the works referred to above seems to be that the robot needs to exhaustively explore the parts of the actuator space it will use. In a group of experiments we showed that it is not necessary to explore the whole actuator space the robot is going to perform in. In paper C we discuss what needs to be trained on and what can be excluded from the training data.

1.2.4 Training the Inverse Model

In the previous section we explained the challenge of choosing which training movements to learn from in order to perform well on novel movements. This section concentrates on how to represent and learn these training movements.

For training the inverse model we chose to adopt the *feedback-error-learning* scheme [35, 38], because it is able to handle redundancies, is a natural extension of a traditional controller, and can be used for control during learning [56]. In addition, it is biologically motivated due to its inspiration from cerebellar motor control [36]. Feedback-error-learning will be further explained in section 2.1.1.

Passold and Stemmer investigated the benefits of applying feedback-error-learning to learn the inverse dynamic model of an INTER scara robot [56]. Experiments were conducted with two types of artificial neural networks (ANNs), a multi-layer perceptron (MLP) and a radial basis neural network (RBF), as the inverse model. The robot's capability to move along a given trajectory by using only a conventional propotional-derivative (PD) controller, a PD controller together with a MLP, a PD controller together with a RBF and a propotional-integral-derivative (PID) controller together with a RBF were compared. The use of an ANN performing in parallel with a conventional controller was advantageous over only using a conventional controller alone, and even though both the MLP and the RBF performed very well, the RBF did it better and faster.

We chose to implement the inverse model with another type of neural network, namely an ESN, because it has been proposed as a cheap and fast architectural and supervised learning scheme and therefore suggested being useful for solving real problems [44]. In addition, ESNs have been associated with how the cerebellum might actually work [85]. Reinhart and Steil used a similar approach for implementing internal models. They trained a recurrent neural network with backpropagation-decorrelation (online version of ESN) to simultaneously learn both the inverse and the forward kinematics model of a redundant robot arm [60]. However, they did not use feedback-error-learning, but calculated the analytical solution and used it directly for training.

The main challenges with applying feedback-error-learning on an ESN are, as explained in paper E, that teacher forcing is not perfect and the feedback error is inaccurate[1]. Even when teacher forcing is very good, there might be stability issues in networks with feedback connections. A classical remedy is adding noise to the internal states during training [29], which makes the network learn the desired next target from the neighborhood of the current state. Other suggestions include using ridge regression [84], pruning the output weights [18] or updating the weights based on the particle swarm optimization algorithm

[1] Teacher forcing is a technique commonly used during training of recurrent networks and means replacing the output of the network with the desired output before the result is fed back in to the network.

[68]. We suggested a new strategy, namely gradually adapting the target output during computation of the output weights. This new method shows quite good results for the feedback-error-learning scheme and is published in the papers D and E.

1.3 Research Questions

The main research objective for this thesis was:

Studying internal model based control schemes and how internal models can be learned and applied for artificial motor control.

In particular, we chose to focus on two important control problems, as motivated in sections 1.2.1 and 1.2.2, resulting in these two research questions:

RQ1: *How can internal models be used to handle delays in the sensory-motor-loop?*
RQ2: *How can internal models be used to make one of the arms coordinate its movements relative to the movements of the other arm?*

As we chose to implement the internal models as ESNs and train the inverse models with feedback-error-learning, the research regarding how to train the internal models can be divided into answering the following two research questions. These were motivated in section 1.2.3 and 1.2.4 respectively.

RQ3: *What characterizes a training movement that makes an ESN generalize to most other movements?*
RQ4: *How can an ESN be trained with feedback-error-learning?*

1.4 Thesis Structure

The thesis is a collection of papers. The research contribution of this thesis is thus constituted of the five included research papers, in their original publication format. The rest of the thesis is organized as follows: The current chapter has given a short introduction to the research topic and motivated the research questions that have been investigated. In the next chapter I will provide more background theory on motor control with internal models and ESNs. The research conducted in the thesis is summarized in chapter 3. The full description of experiments and results can be found in the papers included in part II. Finally, chapter 4 discusses the research methods, limitations and suggested improvements, summarizes contributions, and suggests future work.

1. Introduction

BACKGROUND THEORY 2

In this chapter I will give a short introduction to motor control with internal models and explain the idea behind ESNs based on the theory behind nonlinear modeling and general recurrent neural networks.

2.1 Motor Control with Internal Models

As mentioned in the introduction, there are two types of internal models, *forward* and *inverse models*. These models are illustrated in figure 2.1. In the current context, the forward model predicts the next arm position based on the current position and the motor command, while the inverse model calculates the motor command that will move the arm from the current position to the desired position. Internal models can be used to simulate the *kinematics* and/or *dynamics* of the controlled object or the environment. With kinematics we mean translating the movement trajectory from the task space to the actuator space; that is from an external coordinate system to joint angle configurations for that particular robot. In contrast, dynamics deals with forces, calculating the actual torques that will move the limbs to those joint angles. We have concentrated only on kinematics in this thesis, so when we use the term "motor command", we mean joint angle velocity.

The ability to simulate the kinematics and/or dynamics makes it possible for the controlled object to act proactively. In control theory one would say internal models can be used to realize *feedforward control*, in contrast to *feedback control*, which merely compares a perceived state with a desired state, and uses the difference, or error signal, to adjust the motor command. A feedback controller will try to compensate for the error immediately. How much it compensates depends on the *feedback gain*. Feedback control is also called closed-loop-control because of its tight connection to the sensory signals. An example of a feedback controller is a thermostat that turns on the heat when the temperature drops below the desired value and turns the heat off when the temperature is too high. In contrast, feedforward control uses knowledge about the plant, i.e., the system that is controlled, or the environment to calculate an anticipatory control signal. A feedforward controller could for example turn on the heat when the door is opened, before the temperature drops to the

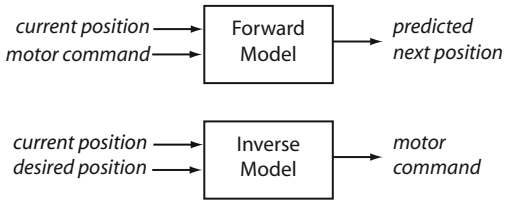


Figure 2.1: A forward model predicts the outcome of a motor command in terms of a position, and an inverse model calculates a motor command that will move the limb to a desired position.

critical point. Feedforward control is sometimes called open-loop-control to emphasize that feedback sensory signals do not directly affect the timing of the response as they do in feedback control. However, that does not mean feedforward control is independent of sensory signals.

In some applications feedback control might work well, but when the feedback loop is slow, the comparison always comes with a delay. Any small error is overcorrected, which will result in even bigger errors, leading to yet bigger corrections. Consequently, relying merely on the sensory feedback when the result of the issued motor command is significantly delayed, will result in highly unstable control [76]. A perfect feedforward controller on the other hand, would perform without error [32]. However, for practical applications it is difficult to generate a perfectly accurate controller. Often a feedback controller is used together with the feedforward controller to compensate for errors and external disturbances.

In control architectures suggested by neuroscientists both feedback controllers and internal models are commonly used. In their investigation into how particularly cerebellum works, several different control architectures have been proposed. For example, it has been suggested that cerebellum implements feedback-error-learning [38, 37], works as a Smith Predictor [50], and consists of multiple pairs of inverse and forward models specialized for different contexts [81]. These suggestions are compared in [82]. In the next subsections we will look further into some of these proposed architectures, which were used as a basis for the control architectures we applied in our experiments. In particular, we will look into three ways to achieve feedforward control with internal models, *feedforward control with an inverse model*, *feedforward control with a forward model* and *forward and inverse models working together*. These three approaches are illustrated in figure 2.2 and explained next.

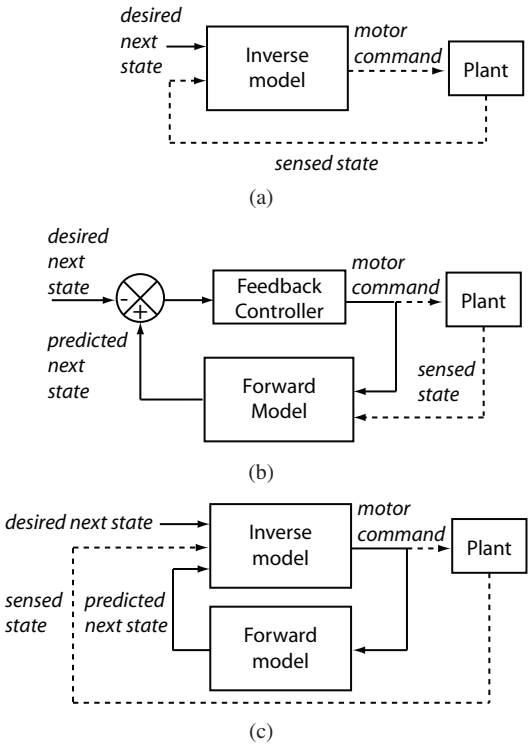


Figure 2.2: Indirect control with internal models can be realized in three ways:

(a) An inverse model is a feedforward controller on its own. In our work the inverse model has a feedback connection from the plant, which is not always the case in other literature.

(b) A forward model coupled with a feedback controller will in principle realize the same function as an inverse model.

(c) There are several ways a forward- and an inverse model can work together to implement feedforward control. Illustrated is the architecture used in paper A.

In all three figures the solid connections indicate fast connections and the dashed indicate slower connections that might result in a problematic delay.

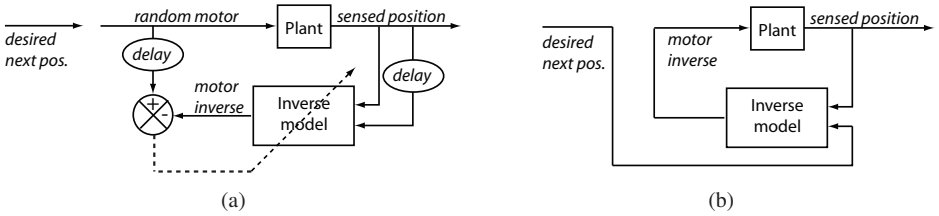


Figure 2.3: *Direct inverse modeling* means trying out motor commands and associate the outcomes with the commands that caused them. Figure (a) shows the architecture used during training, and figure (b) shows the architecture used during testing.

2.1.1 Feedforward Control with an Inverse Model

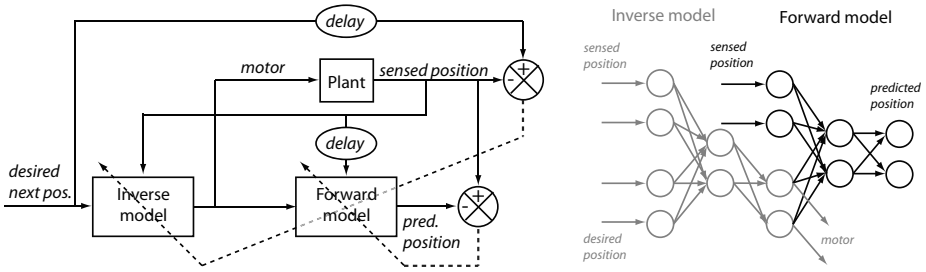
The most straightforward approach for achieving feedforward control is to use only an inverse model, as illustrated in figure 2.2(a). Since the input-output function of the inverse model is ideally the inverse of the body's forward dynamics, an accurate inverse model would produce the desired trajectory that it receives as input perfectly, as long as there are no external perturbations. Acquisition of such an accurate inverse model through learning is, however, problematic, because it requires the desired motor commands to be available, which is generally not possible. What is available is the movement representation in task space.

Three schemas have been suggested for training the inverse model: directly by observing the effect of different motor commands on the plant [35], with a forward model as a distal teacher [33], or with an approach called feedback-error-learning [35, 38]. These will be explained next.

Direct Inverse Modeling

Direct inverse modeling is illustrated in figure 2.3 and amounts to executing motor commands and then associating their outcomes with the commands that caused them. Motor babbling is often used for this purpose, meaning that the motor commands to be executed are chosen at random. Alternatively, one can use a more goal directed approach, e.g., try to execute a specific movement or reach some predetermined end position.

One major drawback with direct inverse modeling is that it needs rewiring before use, which means that it cannot be used for control during learning. Figure 2.3(a) shows the architecture used during training, and figure 2.3(b) shows the architecture used during testing. During training the previous position of the plant together with the current position is used as input to the inverse model, and a motor command is produced. This motor command is compared to the motor command that actually moved the plant from the previous to the current position, and the difference is used to train the inverse model.



(a) (b)

Figure 2.4: In *distal teacher learning* a forward model is learned first. When training the inverse model, the position error is propagated back through the forward model before it is used to train the inverse model. Figure (a) illustrates this architecture, and figure (b) illustrates how the inverse and forward model can be combined into one feed-forward neural network.

During testing, on the other hand, the inverse model is used to control the movement of the plant.

Another drawback pointed out by Jordan and Rumelhart is that most direct inverse learning techniques learn an average of possible actuator space solutions for a given task space goal, this average possibly being an invalid solution [33].

Distal Teacher

In distal teacher learning a forward model is learned first. When training the inverse model, the position error is propagated back through the forward model before it is used to train the inverse model. The architecture is illustrated in figure 2.4(a). The inverse and the forward model can, for example, be implemented as feed-forward neural networks and trained with back-propagation. This makes it easy to combine the two networks into one, as illustrated in 2.4(b). When training the inverse model, the weights belonging to the forward model are kept unaltered.

Feedback-Error-Learning

The feedback-error-learning scheme, illustrated in figure 2.5, relies on the output of a feedback controller that translates the error in position to an error in motor command. This error is then used both to train the inverse model and to adjust the motor command sent to the plant.

We applied feedback-error-learning in most of our experiments.

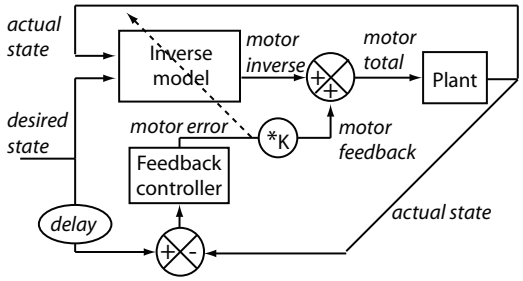


Figure 2.5: *Feedback-error-learning* relies on the output of a feedback controller that translates the position error to an error in motor command. This error is used both to train the inverse model and to adjust the motor command sent to the plant.

2.1.2 Feedforward Control with a Forward Model

A second approach to feedforward control with internal models is to use a forward model together with a feedback controller [50, 48]. As illustrated in figure 2.2(b), the forward model predicts the next state based on the delayed sensed state and the motor command issued by the controller. The idea is that the prediction from the forward model will be available much earlier than the actual sensory feedback, and the controller can react faster by using the difference between this predicted state and the desired state, as opposed to the difference between the sensed state and the desired state. That is, the feedback controller uses internal feedback provided by the forward model instead of external feedback. When the loop through the forward model is fast, this architecture will in practice achieve the same result as an inverse model.

Again, the question is how such a forward model can be acquired through learning. The Smith Predictor provides one solution to the problem of training the forward model online by using a second forward model to compare the prediction made by the first with the actual, delayed, sensed state [50].

We did not use a control architecture without any inverse model in any of our experiments.

2.1.3 Forward and Inverse Models Working Together

As described, model based, indirect feedforward control can be acquired either with the use of an inverse model or by using a forward model. Smagt et al. suggest a third option, namely to use both, a forward and an inverse model coupled together [76]. In their work the forward model was used during actual control, and not just to train the inverse model as in the distal teacher model explained previously.

Wolpert et al. also suggest coupling inverse and forward models together, but their idea

is to use several such pairs, each of which is to be used for different contexts, like when moving in water or moving while carrying something heavy [82]. The role of the forward model in that architecture is to predict which inverse model is the most appropriate for the current situation.

Learning distinct inverse models for different contexts seems like a good idea for robot motor control as well, but we have not come far enough in our research to incorporate this theory. However, we did investigate the benefit of using both a forward and an inverse model together when there are delays in the sensory-motor loop (paper A). The coupled forward-inverse control architecture we used was illustrated in figure 2.2(c).

2.2 Echo State Networks

In all our experiments the internal models were implemented as ESNs. This section describes the idea behind ESNs, how they work, and how they are trained.

An ESN is a recurrent neural network (RNN), which is a neural network with feedback connections as opposed to feedforward networks, which does not contain cycles. Mathematically RNNs implement dynamical systems, while feedforward networks are functions. In theory RNNs can approximate arbitrary dynamical systems with arbitrary precision [13].

RNNs have two obvious theoretical advantages over other methods used for solving temporal tasks. Unlike feedforward neural networks and Support Vector Machines, RNNs have internal memories, and unlike Hidden Markov Models, they can take both discrete and continuous values. In addition, they are of course more biologically plausible. Because of this, they became very popular in the 1980s and 90s. In practice however, supervised training of RNNs with the standard gradient-descent methods are difficult and computationally expensive [42].

In the last years a fundamentally new approach to RNN design and training has attracted new attention to the field. It was proposed independently by Wolfgang Maass under the name of Liquid State Machines (LSMs) [46, 55] and by Herbert Jaeger under the name of Echo State Networks [29, 30, 31]. LSMs and ESMs, together with the more recently explored Backpropagation Decorrelation learning rule for RNNs [69], are given the generic term *reservoir computing* [63, 77].

To understand the idea behind reservoir computing a basic understanding of RNNs and nonlinear modeling are required. I will therefore give a short introduction to these theories in the next two sections, before I use this to explain the general idea behind reservoir computing in section 2.2.3 and specifics about ESNs in section 2.2.4.

2.2.1 Recurrent Neural Networks

A general RNN is illustrated in figure 2.6 and consists of nodes that are tied together by direct connections. There are three types of nodes, input-, internal- and output nodes. The input nodes are not really a part of the RNN, but represent the external input to the network. The output nodes are the nodes presenting the output, and the internal nodes only send information to other nodes inside the network. The state of the input-, internal- and output nodes at time t is denoted as $\mathbf{u}(t)$, $\mathbf{x}(t)$, and $\mathbf{y}(t)$ respectively.

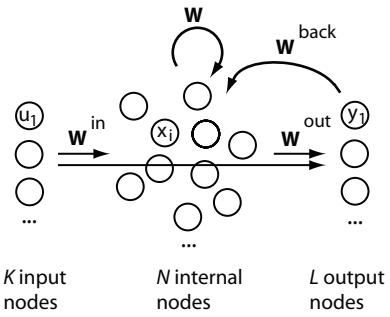


Figure 2.6: The figure illustrates a basic RNN architecture.

The input / internal / output connection weights are collected in $N\text{x}K$ / $N\text{x}N$ / $L\text{x}(K+N)$ weight matrices, $\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{out}$, where K in the number of input nodes, N is the number of internal nodes and L is the number of output nodes. Additionally, the output nodes may project back to the internal nodes with connections whose weights are collected in the $N\text{x}L$ weight matrix $\mathbf{W}^{back}$.

The activation of the internal nodes at time t is updated according to

$$\mathbf{x}(t) = f(\mathbf{W}^{in}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1) + \mathbf{W}^{back}\mathbf{y}(t-1)) + v(t-1), \tag{2.1}$$

where f is the node's activation function, and v is internal noise in the network. The output of the network is computed according to

$$\mathbf{y}(t) = f^{out}(\mathbf{W}^{out}(\mathbf{u}(t), \mathbf{x}(t))). \tag{2.2}$$

These equations can represent all dynamical systems.

2.2.2 Nonlinear Modeling

A general machine learning problem can be defined as a problem of learning a functional relation between a given input $\mathbf{u}(t) \in \mathbb{R}^{N_u}$ and a desired output $\mathbf{y}_{target}(t) \in \mathbb{R}^{N_y}$, where $t = 1, ..., T$, and T is the number of time steps in the training dataset $(\mathbf{u}(t), \mathbf{y}_{target}(t))$.

In a non-temporal task the data points are independent of each other and the goal is to learn a function $\mathbf{y}(t) = f(\mathbf{u}(t))$. In a temporal task the desired output $\mathbf{y}_{target}(t)$ does not only depend on the last input, $\mathbf{u}(n)$, but on a history of previous inputs, i.e., the task is to learn a function $\mathbf{y}(t) = \mathbf{W}(..., \mathbf{u}(t-1), \mathbf{u}(t))$. In both cases the goal is to minimize an error measure $E(\mathbf{y}, \mathbf{y}_{target})$.

For the moment we assume the problem is non-temporal. Many such tasks cannot be accurately solved by a simple linear relation between the input u and the output $\mathbf{y}_{target}$, i.e., a linear model,

$$\mathbf{y}(t) = \mathbf{W}u(t), \tag{2.3}$$

where $\mathbf{W} \in \mathbb{R}^{N_y x N_u}$, gives big errors $E(\mathbf{y}, \mathbf{y}_{target})$ regardless of $\mathbf{W}$.

In such cases a nonlinear models is needed. A number of commonly used methods for nonlinear modeling are based on the idea of nonlinearly expanding the input $\mathbf{u}(t)$ into a high-dimensional feature vector $\mathbf{x}(t) \in \mathbb{R}^{\mathbb{N}}$, and then use linear methods to compute $\mathbf{y}$. The solution can then be expressed by

$$\mathbf{y}(t) = \mathbf{W}\mathbf{x}(t) = \mathbf{W}\mathbf{x}(\mathbf{u}(t)), \tag{2.4}$$

where $W \in \mathbb{R}^{N_x x N_u}$. Finding $\mathbf{W}$ is a well defined and understood problem, but producing a good expansion function, x, generally involves more creativity.

One such nonlinear method is *Support Vector Machines*. Here, the function $\mathbf{x}(t)$ is called *kernel* [47]. Using feedforward neural networks is another method. A feedforward network with one hidden layer computes

$$\mathbf{y}(t) = f_{out}(\mathbf{W}^{out}\mathbf{x}(\mathbf{u}(t))), \tag{2.5}$$

$$\mathbf{x}(t) = f_{in}(\mathbf{W}^{in}\mathbf{u}(t)). \tag{2.6}$$

The same idea is used for temporal tasks, but the expansion function must now have memory, i.e., $\mathbf{x}(t) = \mathbf{x}(\mathbf{x}(t-1), \mathbf{u}(t))$, as in equation 2.1.

The classical approach to supervised training of neural networks is *gradient-decent*. It iteratively adapts all weights $\mathbf{W}^{out}$, $\mathbf{W}$, $\mathbf{W}^{in}$ and possibly $\mathbf{W}^{back}$ according to their estimated gradient $\frac{\delta E}{\delta \mathbf{W}^{all}}$ in order to minimize the error $E(\mathbf{y}, \mathbf{y}_{target})$. Different classical gradient-decent methods are presented in [2, 57].

2.2.3 The Idea behind Reservoir Computing

Reservoir computing methods differ from the traditional gradient-decent methods by conceptually and computationally separating the expansion function, x and the readout, y. Training/generating them separately and even with different goal functions makes sense because they serve different purposes.

The nonlinear, temporal expansion function is called the *reservoir*, and it is usually implemented as a recurrent neural network. This reservoir is generated randomly and remains unchanged during training[1]. Its function resembles a tank of liquid. One can think of the input as stones thrown into the liquid, creating unique ripples that propagate, interact and eventually fade away. After learning how to read the waters surface, one can extract a lot of information about recent events without having to do the complex input integration. Real water has successfully been used as reservoir in such a manner [20].

The readout is essentially non-temporal and can be generated as a linear combination of the signals from the reservoir. Learning this function is typically quick, which makes these methods computationally efficient compared to gradient decent methods.

2.2.4 Training Echo State Networks

The task is described by a set of input and desired output pairs, $[\langle \mathbf{u}(1), \mathbf{y}_{target}(1)\rangle, \langle \mathbf{u}(2), \mathbf{y}_{target}(2)\rangle, ..., \langle \mathbf{u}(T), \mathbf{y}_{target}(T)\rangle]$, and the solution is a trained ESN whose output $\mathbf{y}(t)$ approximates the desired output $\mathbf{y}_{target}(t)$, when the ESN is driven by the training input $\mathbf{u}(t)$. The error function to be minimized is the normalized root-mean-squeare error (NRMSE), which is the root-mean-square error (RMSE) divided by the range of the possible target values:

$$E(\mathbf{y}, \mathbf{y}_{target}) = \frac{\sqrt{\| \mathbf{y}(n) - \mathbf{y}_{target}(n) \|^2}}{y_{max} - y_{min}}. \tag{2.7}$$

The ESN is generated in three steps:

Step 1: Provide a random ESN

Initially, a random RNN with the Echo State property is generated [30]. Informally, the echo state property says that if the network runs for a very long time, the network state will be uniquely determined by the history of the input and the output.

Two important parameters for creating the random ESN is the network size, N, which should reflect both the length of the training data and the difficulty of the task, and the spectral radius, α, which determines the length of the memory of the network. We use $tanh$ as the activation function, which means that $\alpha \in [0, 1]$. In our work these parameters have been found by trial and error for each experimental setup as recommended by Jaeger [29].

The result of this first step is the initial weight matrixes $\mathbf{W}^{in}$, $\mathbf{W}$, $\mathbf{W}^{out}$ and $\mathbf{W}^{back}$.

[1] Lately, quite some research is done on altering the reservoirs to improve performance on a given application, but we stick to the original idea of keeping the reservoir fixed.

Step 2: Harvest the states of the nodes in the network

Using the initial weight matrixes, the network is driven by the provided input sequence, $[\mathbf{u}(1), \mathbf{u}(2), ...\mathbf{u}(T)]$. When there are feedback connections from the output nodes to the internal nodes, teacher forcing is used, meaning $\mathbf{y}_{target}(t)$ is used instead of the actual output, $\mathbf{y}(t)$, when computing $\mathbf{x}(t+1)$ and $\mathbf{y}(t+1)$.

The first T_0 time steps are used to wash out the initial transient dynamics. After this initialization, the state of each input- and internal node in each time step is stored in a state collection matrix, **M**. In the end, **M** has the dimension $T \times (K+N)$. Just to remind, K, N and L are the number of input-, internal- and output nodes respectively. Assuming $tanh$ is used as output activation function, $tanh^{-1}\mathbf{y}_{target}(t)$ is collected row-wise into a target collection matrix, resulting in a matrix **S** of size $T \times L$.

Step 3: Compute the output weights

Equation 2.2 can now be written:

$$\mathbf{S} = \mathbf{M}(\mathbf{W}^{out})^T. \tag{2.8}$$

The goal is to solve this equation with regard to $\mathbf{W}^{out}$.

There are two possible problems when solving this equation: 1) the equation may not have any solution, or 2) the equation may have many solutions. The Moore-Penrose pseudo-inverse will in the first case provide the least squares solution, and in the second case provide the solution with the minimum Euclidean norm. We used this pseudo-inverse to calculate the output weights:

$$(\mathbf{W}^{out})^T = \mathbf{M}^{+}\mathbf{S}. \tag{2.9}$$

2.2.5 Challenges in Reservoir Computing

Besides the application studies, the bulk of current research on reservoir computing is devoted to optimal reservoir design, or reservoir optimization algorithms. The general "no free lunch" principle in supervised machine learning states that there can exist no bias of a model that would universally improve the accuracy of the model for all possible problems [83]. In the current context this means that no single type of reservoir can be optimal for all types of problems. A detailed review of reservoir optimization strategies can be found in [45].

Another challenge is the stability issue when the network has feedback connections from the output layer [29]. This problem is discussed further in paper D, as it is related to our problems with training the inverse model with feedback-error-learning.

RESEARCH SUMMARY 3

This chapter summarizes the research conducted in the thesis. First, an overview of the material and the employed methods and research process is given. Then, a list of the included papers follows, and finally, each paper is presented.

3.1 Overview

The overall objective of the thesis is, as mentioned in the introduction, to *study internal model based control schemes and how internal models can be learned and applied for artificial motor control*. To do this we developed a simple, lightweight simulator we call Skinny, which learns internal models of his motor apparatus by imitating the human motion.

The human motion was recorded from arm movements of a person, and the input to Skinny's control system is a sequence of the recorded positions of hands and elbows relative to the shoulders. The task of Skinny's control system is to generate motor commands that result in Skinny imitating the recorded movement. This is solved by learning internal models of how the arms work. Figure 3.1 illustrates the setup.

We chose to implement the internal models as ESNs as it has been proposed as a cheap and fast architectural and supervised learning scheme and is therefore suggested being useful

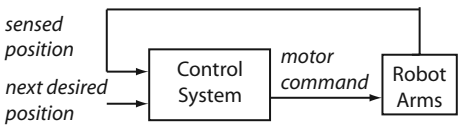
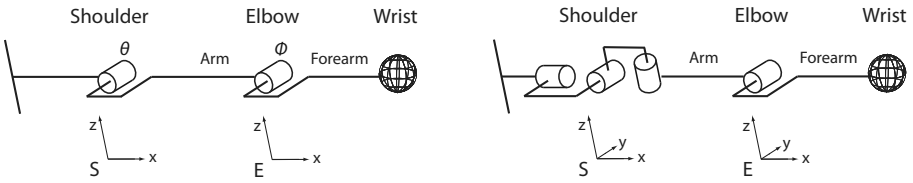


Figure 3.1: The input to the control system is the desired next position of the robot arms. This is used together with the sensed position of the arms, and the output is a motor command. The motor command moves the robot's arms.



(a) Joint representation in 2D

(b) Joint representation in 3D

Figure 3.2: The motor commands are the joint angle velocities.

for solving real problems [44]. We chose to train the inverse model with feedback-error-learning because it is a natural extension of a traditional feedback controller and can be used for redundant systems [56]. Another advantage with feedback-error-learning is that it can be used for control during learning, but we did not exploit this.

3.1.1 The Simulator

All the experiments were done with the simple robot simulator Skinny. Initially we tried to apply the multi-agent physics simulator Breve [39], but running the experiments trough Breve was very time consuming, and in any case we wanted to start with simple kinematic control, and thus did not need the full functionality of Breve.

As we chose to start simple by only concentrating on kinematic control, Skinny's limbs are directly controlled by the joint angle velocities, and we call them the motor commands. However, we believe our methods can be used also for dynamic control.

Skinny can operate in both 2 and 3 dimensions. In 2D he has 4 degrees-of-freedom (DOFs), one for each shoulder and one for each elbow. In 3D he has two additional DOFs in each shoulder, giving him a total of 8 DOFs. This is illustrated in figure 3.2.

The range of motion was constrained to be between 0^o and 180^o for all 4 DOFs, and if the motor command implies moving the limb further, the limb stops at the limit and the overshooting motor command is ignored.

In some of the experiments the maximal joint angle velocity was limited to 50-100% larger than the maximum velocity registered in the recorded movement. This meant that a joint angle velocity equal to 1 moved the joint far less than 180 degrees. Limiting joint velocity is realistic because no robot can move its limbs arbitrarily fast. It also makes large errors in motor commands lead to smaller position errors.

Skinny, the ESNs, and all the experiments were implemented in matlab, and we used the matlab toolbox for ESNs written by Herbert Jaeger et al. [19].

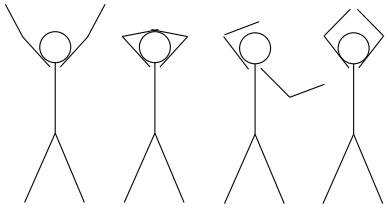


Figure 3.3: The YMCA movement.

3.1.2 The Dataset

In most of our experiments we used the YMCA dataset, which is a recording of the dance to the song YMCA by the Village People. The movement is illustrated in figure 3.3. This dataset was used because the movement is well known and easy to explain and recognize. At the same time it is complex enough to be interesting. One drawback of the YMCA movement is the symmetry of the *Y*, *M* and *A* movements; only the *C* movement breaks this symmetry.

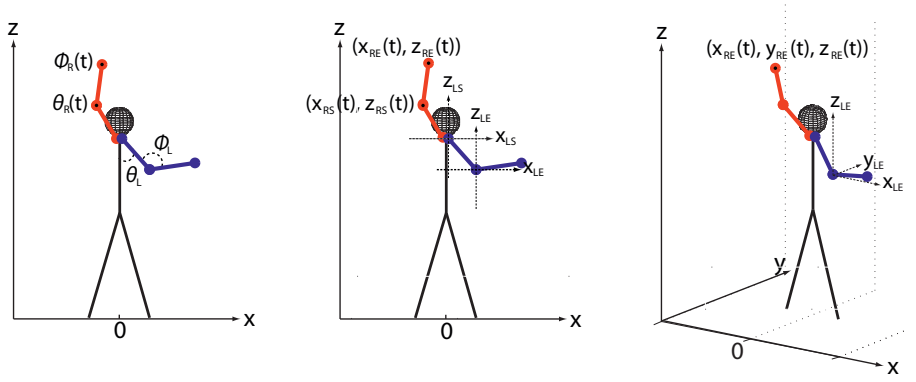
The movement data was gathered with a Pro Reflex 3D motion tracking system by Axel Tidemann [71]. The system was able to track the position of fluorescent balls within a certain volume by using five infrared cameras. The sampling frequency of the Pro Reflex was 200 Hz. In the experiments we mostly used only every fourth sample, meaning the position trajectory consisted of 50 samples/sec, resulting in a 313 steps long sequence. The tracking of the balls yields Cartesian coordinates of the balls, and we used this to make three different representations of the recorded arm position as illustrated in figure 3.4 and explained next.

Positions in 2D joint angles

The movement trajectory was projected down into 2D and translated into joint angles which were normalized to the interval $\langle -1, 1 \rangle$. The result is a position sequence with four values per time step, the elbow and shoulder angles in 2D.

Positions in 2D Cartesian coordinates

The 2D projection was used as the position trajectory, and consisted of the x and z coordinates of the elbow relative to the shoulder and the wrist relative to the elbow. The coordinates were normalized to be in the interval $\langle -1, 1 \rangle$. The position in each time step was thus represented by 8 signals.



(a) Position in 2D joint angles. (b) Position in 2D Cartesian coordinates (c) Position in 3D Cartesian coordinates

Figure 3.4: Position representation.

Positions in 3D Cartesian coordinates

The positions were represented in x, y and z coordinates of the wrist relative to the elbow and the elbow relative to the shoulder. The coordinates were as usual normalized to be in the interval $\langle -1, 1 \rangle$. This means that each position was represented by 12 signals.

3.1.3 The Kinematic Control Problem

As was illustrated in figure 3.1, the task for the control system was to calculate the joint angle velocities that will keep the arms on the desired trajectory. The different position representations and modalities lead to control problems with various levels of complexity.

Linear Control Problem

In the simplest case Skinny moves in 2D and the arm positions are represented in joint angles. The motor commands are angle velocities, thus, the control system must learn to calculate the linear transformation from joint angles to joint angle velocities.

Nonlinear Control Problem

Skinny still moves in two dimensions, but the arm positions are now represented as Cartesian coordinates in this 2D space. The control system will need to learn the nonlinear transformation from Cartesian coordinates to joint angle velocities.

Nonlinear Control Problem with Many Solutions

When Skinny is represented in 3D with positions in Cartesian coordinates, the inverse problem becomes a one to many mapping, as several joint angle configurations corresponds to the same arm position in Cartesian coordinates.

The analytical solution used for training in this setup is the inverse kinematics approximation proposed by Tolani and Badler [72].

3.1.4 Research Progress

In the introduction we stated four research questions:

RQ1: *How can internal models be used to handle delays in the sensory-motor-loop?*
RQ2: *How can internal models be used to make one of the arms coordinate its movements relative to the movements of the other arm?*
RQ3: *What characterizes a training movement that makes an ESN generalize to most other movements?*
RQ4: *How can an ESN be trained with feedback-error-learning?*

The initial focus was on investigating RQ1 and RQ2, namely how to use internal models to handle delays in the sensory-motor-loop and to realize bimanual coordination. The results of this work are discussed in paper A and B respectively.

During the initial work on handling delays and bimanual coordination we realized we had to spend more time investigating how to learn the internal models, particularly the inverse model. In paper C we studied the generalization capabilities of the inverse model by letting Skinny train on imitating one movement and tested what other movement he could imitate without further training. This paper includes our contributions to RQ3.

As we moved on from the linear control problem to the nonlinear control problem and the nonlinear control problem with many solutions, we discovered that just learning the training movements was a challenge in its own. Our last two papers, paper D and E, address this problem by investigating the training of an ESN with feedback-error-learning, i.e., RQ4. This research resulted in a novel training method for ESNs.

3.2 List of Publications

Papers Included in Thesis

A Rikke Amilde Løvlid and Pinar Öztürk. Dancing YMCA with delayed sensory feedback. In *Proceedings of the Tenth IASTED International Conference on Artificial Intelligence and Applications*, 2010.

B Rikke Amilde Løvlid and Pinar Öztürk. Learning bimanual coordination patterns for rhythmic movements. In Konstantinos Diamantaras, Wlodek Duch, and LazarosS. Iliadis, editors, *Artificial Neural Networks ICANN 2010*, volume 6354 of Lecture Notes in Computer Science, pages 143–148. Springer Berlin Heidelberg, 2010.

C Rikke Amilde Løvlid. Learning motor control by dancing YMCA. In A*rtificial Intelligence in Theory and Practice III*. 2010.

D Rikke Amilde Løvlid. Learning to imitate YMCA with an ESN. In A*rtificial Neural Networks and Machine Learning - ICANN 2012*, Lecture Notes in Computer Science, pages 507–514. Springer Berlin Heidelberg, 2012.

E Rikke Amilde Løvlid. A novel method for training an echo state network with feedback error learning. *Advances in Artificial Intelligence*, 2013.

Other Papers

F Rikke Amilde Løvlid. Introducing learning rate analogy to the training of echo state networks. In *First Norwegian Artificial Intelligence Symposium*, 2009.

3.3 Paper Descriptions

This section summarizes the papers constituting the thesis, including motivation, paper abstract and a summary of experiments and important results.

3.3.1 Paper A - Dancing YMCA with Delayed Sensory Feedback

The first paper addresses the problem of handling delays in the sensory-motor-loop. Humans perceive their own movement with a delay. The time elapsed for the travel of neural signals from the brain to the muscles and the time the brain needs to process the sensory feedback it receives constitute problems for the smoothness of voluntary movement.

The delay issue poses a problem in artifacts as well. Robots' electrical circuits do not suffer from the same conduction delays as the neurons in the human nervous system, but even if the delay is much smaller, a robot may need to move so fast that it becomes a problem. Also, depending on the complexity of the robot's perceptual capabilities, it might take time to process the sensory signals, e.g., to interpret visual sensory input.

In this paper we compared two control architectures, one with only an inverse model and one with an inverse model and a forward model coupled together.

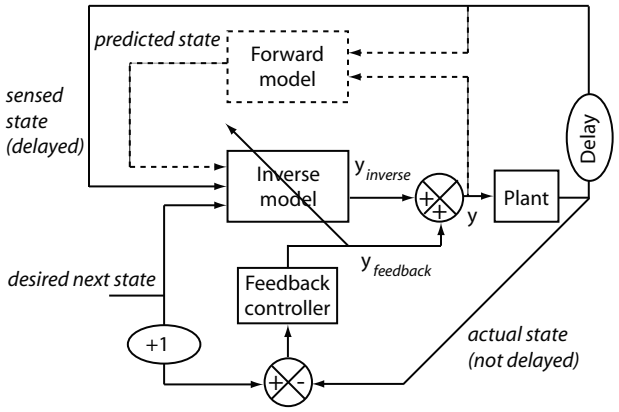


Figure 3.5: Two control architectures, one has only an inverse model while in the other, the inverse model is coupled with a forward model. The dotted loop including the forward model is only enabled in the experiments where the forward model is involved. Delays are shown with ellipses (+1 means delayed with one time step), an arrow across a model represents training, and $\mathbf{y}$ is the motor command.

Abstract: *Lack of sensory feedback or delay in feedback has been shown to have detrimental effects on cognition and action in humans and in artifacts. Despite the adverse effects of delay, humans manage to generate smooth and timely movements. This has been explained by the existence of predictive models in the brain. In this paper we investigate the possible role of a predictive model that anticipates the consequences of the motor command to be issued to the actuator (e.g., arm). The paper presents two architectures, one with and the other without predictive components, and compares their performances in dancing to the song 'YMCA'. The architecture including the predictive model has been trained in three different ways to uncover the possible effects of the training method on the movement performance. The results confirm the role of prediction in the movement control.*

Experiments

In a group of experiments we added delay in Skinny's sensory-motor loop and compared the performance of two control architectures. The first had only an inverse model and the other had both an inverse and a forward model working together. The architectures are illustrated together in figure 3.5 and did both include a conventional feedback controller as in the feedback-error-learning architecture.

The architectures were tested on the linear control problem from section 3.1.3, and the

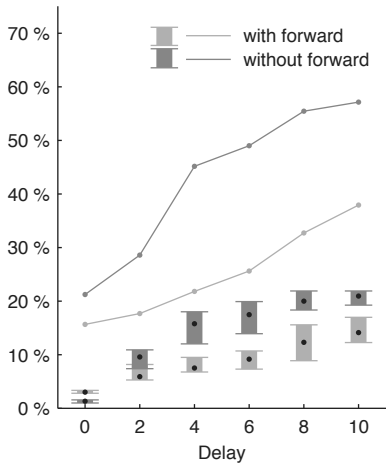


Figure 3.6: The figure shows a comparison of the performance with and without a forward model with delayed sensory input. The line plots illustrate the online correction ratio$y_{feedback}/y$, witch is the feedback controller's contribution to the total motor command in percent.

experiments were run 10 times with each architecture. To evaluate the performance, we compared the difference between the desired and actual state averaged over all time steps in the last epoch. Also the average online correction ratio, $y_{feedback}/y$, was compared. This ratio reflects the feedback controller's contribution to the total motor command in percent. The feedback gain was 0.65 throughout the experiments.

Results

The results are plotted in figure 3.6. For delay ≥ 4 the performance error in the worst run with a forward model was better than the performance error in the best run without, which strongly suggests adding a forward model is beneficial when sensory input is delayed. In fact, the results indicate that the combined architecture, with both a forward and an inverse model, also performs better when there is no delay.

Figure 3.6 also shows that the feedback controller was less needed when the forward model was used, as the online correction ratio was lower.

In the experiments, whose results are plotted in figure 3.6, the forward model was pretrained on the training movement and kept fixed during the training of the inverse model. We also tried training the two models in parallel. Best results were gained when pretraining the forward model, but the difference became less evident as the delay increased.

The main challenge when investigating the use of a forward model to compensate for delayed sensory input was how to combine the delayed, actual position with the predicted position to make a good estimate of the position. The simplest solution was not to precompile the two signals at all, but use both the prediction from the forward model and the delayed sensory information as separate inputs to the inverse model together with the desired next state. We got satisfactory results with this simple solution and therefore did not proceed to incorporate more sophisticated methods like applying a Kalman filter.

3.3.2 Paper B - Learning Bimanual Coordination Patterns for Rhythmic Movements

Coordinated bimanual movements form the basis in many everyday motor skills. In human bimanual coordination there are several basic principles or default coordination modes, such as the preference for in-phase or anti-phase movements, e.g., the two arms mirroring each other or making the opposite movement.

In this paper we studied an artificial system that learned bimanual coordination patterns with various phase differences, frequency ratios and amplitudes.

Abstract: *Coordinated bimanual movements form the basis for many everyday motor skills. In human bimanual coordination there are several basic principles or default coordination modes, such as the preference for in-phase or anti-phase movements. The objective of our work is to make robots learn bimanual coordination in a way that they can produce variations of the learned movements without further training. In this paper we study an artificial system that learns bimanual coordination patterns with various phase differences, frequency ratios and amplitudes. The results allow us to speculate that when the relationship between the two arms is easy to represent, the system is able to preserve this relationship when the speed of the movement changes.*

Experiments

In this paper we investigated how Skinny can learn to move one arm, the lagging arm, in a specific pattern relative to the other, the leading arm. Our work was inspired by the crosstalk-hypothesis, which states that there is a connection between the motor systems of the two coordinated limbs [3, 70, 12].

The control architecture of the lagging arm is illustrated in figure 3.7. The movement of the leading arm was forced. During testing the velocity of the leading arm was changed, and we observed whether or not the lagging arm was able to adapt to this new velocity and keep the pattern it had trained on.

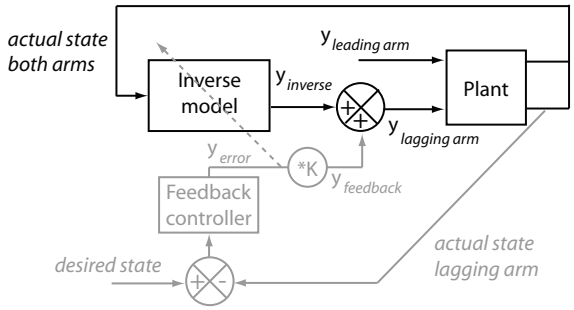


Figure 3.7: The bimanual control architecture. An arrow represents training, and u is a motor command.

Three movement properties where used to characterize and analyze the affordance of he bimanual architecture: relative phase, frequency and amplitude.

Results

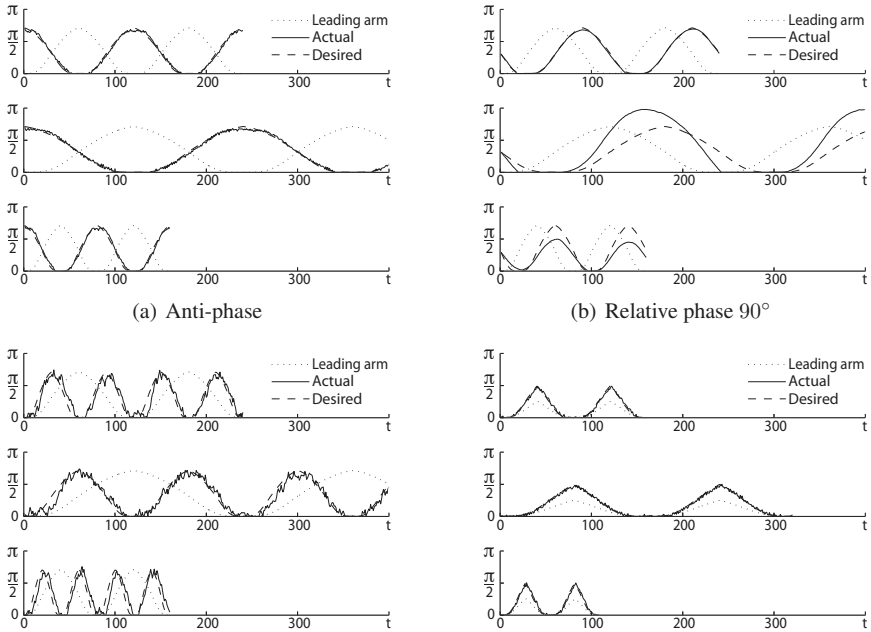
As it is with humans, in-phase and anti-phase movements were more stable than the movements with other phase relationships. As for frequency ratio, polyrhythms like 2:3, appeared to be impossible to learn, whereas simpler rhythms like 1:2 and 1:3 seemed relatively stable once learned. The control system had no problem generalizing when it was trained with different amplitudes on the two arms. The results are illustrated in figure 3.8.

These results allowed us to speculate that when the relationship between the two movement components is easy to represent, the system is able to preserve this relationship when the speed of the movement changes. Further work remains to formally define the characteristics of the movement that are easier to coordinate, and what is easy to represent.

3.3.3 Paper C - Learning Motor Control by Dancing YMCA

Skinny starts without any knowledge about how his motor apparatus works. Before he can imitate anything, the internal models must be learned. In this paper we investigated whether Skinny could learn to imitate a group of "test" movements after training on only one. This involves the study of the generalization capabilities of the ESN.

Abstract: *To be able to generate desired movements a robot needs to learn which motor commands move the limb from one position to another. We argue that learning by imitation might be an efficient way to acquire such a function, and investigate favorable properties of the movement used during training in order to maximize the control system's generalization capabilities. Our*



(a) Anti-phase

(b) Relative phase 90°

(c) Frequency ratio 1:2

(d) Different amplitude

Figure 3.8: The bimanual experiments were done with the 2D simulator with positions in joint angles. The figures show the results when the trained control system was tested on the same movement as it trained on in the original speed (top), half the original speed (middle), and with 50% increase in the original speed (bottom). The shoulder angle of the leading arm and the desired and actual shoulder angle of the lagging arm at each time step are plotted. The movement of the leading arm is shown together with the desired and actual movement of the controlled lagging arm.

(a) The generalization to different speeds is close to perfect for the anti-phase movement.

(b) The phase difference of 90° seems harder to preserve when speed is changed.

(c) The system was trained to perform the basic movement with frequency ratio 1:2. The movement of the lagging arm is not smooth, but the timing is just as good in the testing case as in the training case.

(d) The lagging arm moves twice as far within the same time interval as the leading arm. The control system has no problem generalizing.

control system was trained to imitate one particular movement and then tested to see if it can imitate other movements without further training.

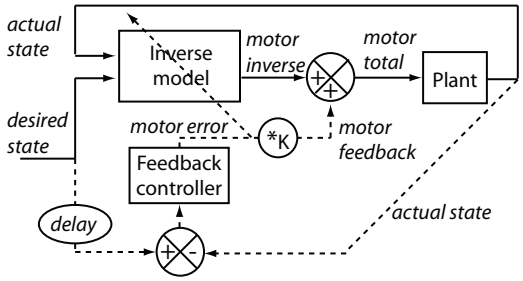


Figure 3.9: The feedback-error-learning architecture used for testing generalization capabilities. An arrow represents training, and $\mathbf{y}$ is a motor command. The feedback gain, K, was linearly decreased from 1 to 0 during training and kept 0 during testing.

Experiments

The experiments were done with the basic feedback-error-learning architecture, as illustrated in figure 3.9. The feedback gain was linearly decreased from 1 to 0 during training, and in order to test the capabilities of the inverse model, the feedback controller was not used during testing. To generate different training and testing sequences, the YMCA movement was manipulated in various ways, and the network was trained on one movement and tested on other movements.

Results

First, we verified that when trained to imitate one movement, the control system had not only learned to imitate that particular movement, but was able to imitate novel movements without further training. This meant that the system had learned at least parts of the desired inverse model.

Second, we showed that in order to learn to control one particular degree of freedom, it has to be included in the training movement. In addition, our results suggest that the control system does not have to train on the whole range of motion for each degree of freedom in order to generalize to all movements. Figure 3.10 illustrates this. Not having to train on the full range of motion is important when we want to train the inverse model with minimal amount of effort.

Third, asynchronous movements proved to be harder than synchronous movements. The control system was not able to produce different motor commands for the two arms when it had not been trained to do so. For humans it is indeed true that it is easier to move the limbs synchronously. It is still very interesting that we get the same results for this control system. It is also interesting to see that a system trained to produce a certain synchronous movement, when asked to generate an asynchronous movement, provides

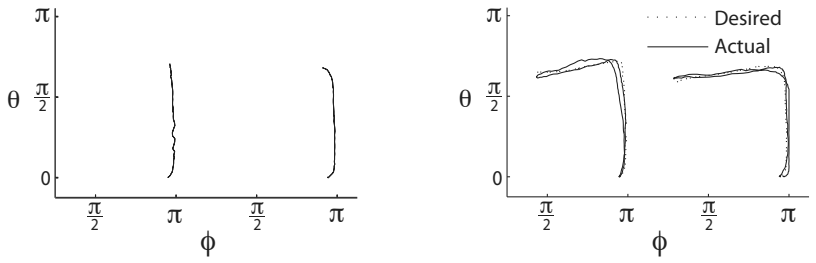


Figure 3.10: The left figure illustrates the *Y* part of the YMCA movement, where ϕ is the elbow angle and θ the angle in the shoulder. Notice that there is hardly any motion in the elbow. The figure to the right shows the result when the control system trained on movement *Y* is tested on movement *YM*. The control system is able to generate more motion in the elbow joints than it trained on, and the imitation of YM is near perfect.

the average between the desired movement of the left and right arm, which is the best possible solution the system could provide.

In summary, our findings suggest that imitation may be used as an efficient method to learn the inverse model, because one can choose the training sequence optimally, as opposed to exploration without guidance. This conclusion is supported by Rolf et al., who suggest the use of goal directed exploration in contrast to motor babbling [61].

3.3.4 Paper D - Learning to Imitate YMCA with an Echo State Network

As mentioned, we implemented the internal models as ESNs. Training an ESN to learn the inverse model proved to be rather challenging. We tried to use feedback-error-learning, but it did not work well together with the standard method for training ESNs, which was explained in section 2.2.4. This led to a proposal of a novel training method for ESNs. This paper compares the new training method with the original, standard method.

Abstract: *When an echo state network with feedback connections is trained with teacher forcing and later run in free mode, one often gets problems with stability. In this paper an echo state network is trained to execute an arm movement. A sequence with the desired coordinates of the limbs in each time step is provided to the network together with the current limb coordinates. The network must find the appropriate angle velocities that will keep the arms on this trajectory. The current limb coordinates are indirect feedback from the motor output via the simulator. We do get a problem with stability in this setup. One simple remedy is adding noise to the internal states of the network. We verify that this helps, but we also suggest a new training strategy that leads*

to even better performance on this task.

The Novel Training Method

In the original training algorithm the training sequence is run through the network once, and the output weights are updated based on the target and the internal states of the network as given in equation 2.9. We suggest running the training sequence through the network several times. In each cycle the weights are calculated based on the internal states and something in between the estimated target and the actual output from the inverse model. The target used when computing $\mathbf{W}^{out}$ in cycle i is

$$\mathbf{y}^i_{used\ target}(t) = \beta \mathbf{y}_{estimated\ target}(t) + (1-\beta)\mathbf{y}_{ESN}(t), \tag{3.1}$$

where

$$\mathbf{y}_{estimated\ target}(t) = \mathbf{y}_{ESN}(t) + \mathbf{y}_{error}(t+1). \tag{3.2}$$

Note that when $\beta = 1$, we get the original training method, where $\mathbf{y}^i_{used\ target}(t) = \mathbf{y}_{estimated\ target}(t)$.

Experiments

In this paper the performance of the new training method was compared to the performance of the original training method, both when the inverse model was trained with feedback-error-learning, and when it was trained on the analytically calculated true target, $\mathbf{y}_{true\ target}$. The two architectures are illustrated in figure 3.11.

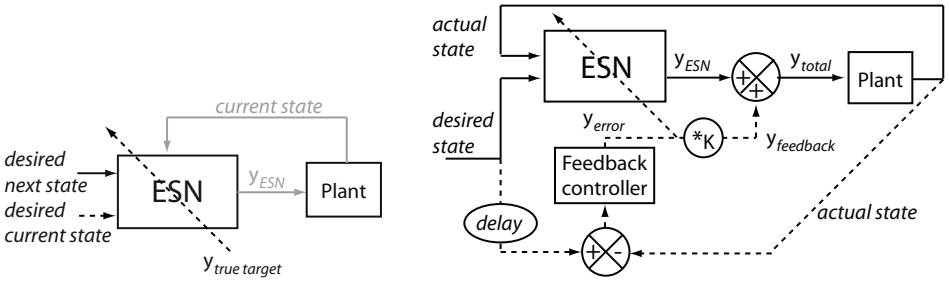
We attempted to improve the performance of the original training method by adding noise to the internal network and using a longer training sequence by repeating the movement multiple times. Internal noise was also added when using the new training method, but only one repetition of the training movement was used in the training sequence when applying the new method.

To evaluate the results, we use the Root Mean Square Error (RMSE) normalized over the range of the output values[1],

$$NRMSE(\mathbf{y}, \mathbf{y}_{true\ target}) = \frac{\sqrt{MSE}}{y_{max} - y_{min}} = \frac{1}{2}\sqrt{\frac{\sum_{i=0}^{n}(y^i_{true\ target}(t) - y^i(t))^2}{n}} \tag{3.3}$$

The NRMSE for each run was averaged over all time steps and DOFs.

[1]The RMSE can be normalized in different ways, all called NRMSE. We used the absolute difference between the smallest and the largest possible value. One could also use the range of the actually observed data or the average of the observed data. Several researches on ESNs normalize over the standard deviation of the target output [43, 84].



(a) True Target (b) Feedback Controller

Figure 3.11: Two architectures were used in the experiments: (a) the inverse model is trained with the true target information and teacher forcing, and (b) a feedback controller is used both for estimating the motor error and for providing teacher forcing. In the latter, the feedback gain, K, was gradually decreased from 1 to 0 during several rounds of training. The dotted lines are only used during training whereas the grey lines are only used during testing.

Results

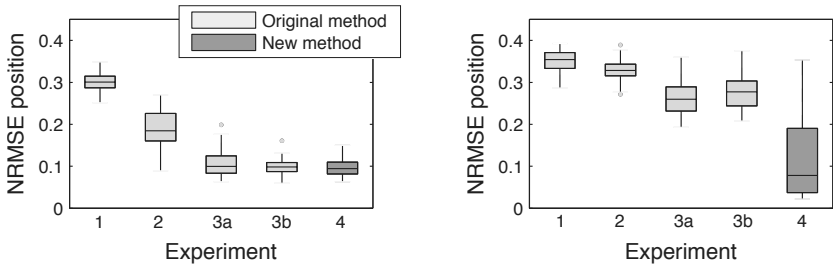
Figure 3.12 illustrates the performance of the new training method versus the original when the ESN is trained with the true target (figure 3.12(a)) and feedback-error-learning (figure 3.12(b)). The new method shows particularly good results when the ESN must be trained with feedback-error-learning. The performance in the true target setup seems equivalent; however, it remains to be tested which method is faster.

3.3.5 Paper E - A Novel Method for Training an Echo State Network with Feedback Error Learning

The novel training method is studied further in this last paper which focus on how and why it works so well when the ESN is trained with feedback-error-learning.

Abstract: *Echo state networks are a relatively new type of recurrent neural net- works which have shown great potentials for solving nonlinear, temporal problems. The basic idea is to transform the low dimensional temporal in- put into a higher dimensional state, and then train the output connection weights to make the system output the desired information. Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks.*

This paper investigates using an echo state network to learn the inverse kinematics model of a robot simulator with feedback-error-learning. In this scheme teacher forcing is not perfect, and joint constraints on the simulator



(a) True Target (b) Feedback Controller

Figure 3.12: Box and whisker plot for 50 runs of each of following experiments: 1) *Original training method without noise during training.* 2) *Original training method with noise in the network.* 3) *Original training method with noise and several repetitions of the YMCA movement in the training sequence. The experiments when using 5 and 10 repetitions are referred to as 3a and 3b respectively.* 4) *New proposed training method with* $\beta = 0.1$. *There is still noise in the network, and the training sequence consists of only one repetition of the YMCA movement.*

On each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually.

make the feedback error inaccurate. A novel training method which is less influenced by the noise in the training data is proposed and compared to the traditional ESN training method.

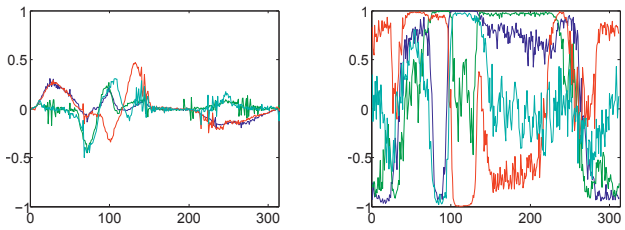
Experiments

All the experiments in this paper use the feedback-error-learning architecture, and internal noise is included both with the original and the new training method. The optimal amount of internal noise is tested, and it appears this training method requires more noise in the reservoir than the original method. In this paper we also show the results for different values of β.

In addition to the above experiments, which continue the experiments in paper D, we investigated further why the new method works better than the original. To do this we trained the same initial network with A) *the original method without repeating the training movement*, B) *the original method with the training movement repeated 5 times*, and C) *the new method with* $\beta = 0.1$. The results are presented next.

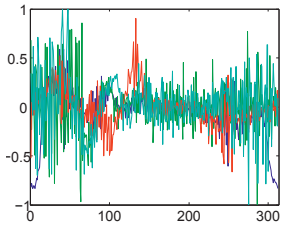
Results

Figure 3.13 shows why experiment A fails. The estimated target sequence is too noisy, and with the short training sequence without any repetitions, the output from the ESN

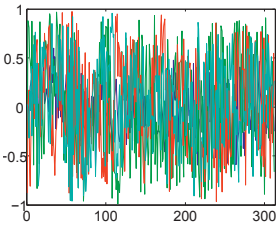


(a) True target

(b) Initial ESN output



(c) Estimated target



(d) ESN output after trained with (c)

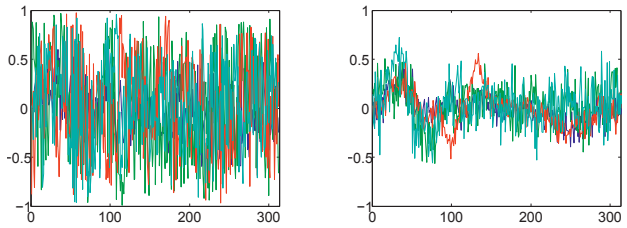
Figure 3.13: The plots illustrate why the original method without repetitions (experiment A) fails. Compared to the true target, (a), the estimated target in the first epoch, (c), is very noisy. It has the general shape of the true target, but when training the initial, random ESN, (b), with this noisy estimate, the result is a network which outputs mostly noise, (d). This only gets worse in the succeeding epochs. Plotted are motor commands (joint angle velocities) for the 4 DOFs at each time step in the training sequence.

becomes even noisier.

The output from the ESN after training becomes significantly less noisy when the movement is repeated several times in the training sequence, as illustrated in figure 3.14. In this setup the target sequence has a repeating pattern, and since the error in each repetition will differ, the weight calculation will average over these slightly different representations.

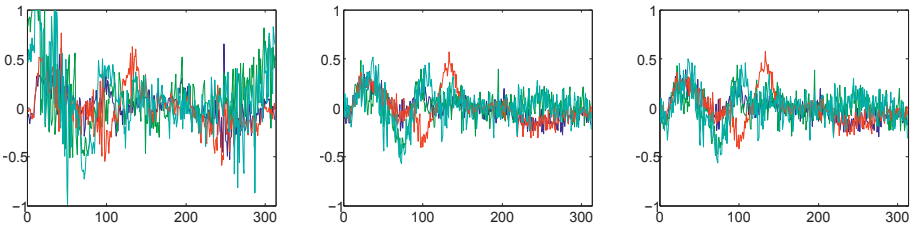
When using the new training method, the approach for making a smoother target is different. The new method is apparently able to keep the smoothness of the output of the first, random network, and just gradually drive that solution toward the target. As illustrated in figure 3.15, the used target, i.e., the best target estimate combined with the previous ESN output, appears much less noisy than the target estimate alone.

The new method also results in better teacher forcing. Figure 3.16 illustrates the quality of the teacher forcing for the three selected experiments.



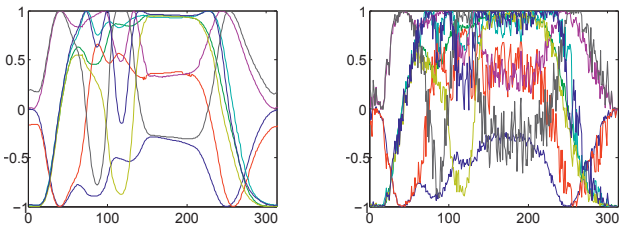
(a) (b)

Figure 3.14: Adding more repetitions of the movement in the training sequence makes the output of the ESN seem less noisy. Plot (a) shows the output of the ESN after training with one repetition and plot (b) the ESN output after training on 5 repetition of the YMCA movement.



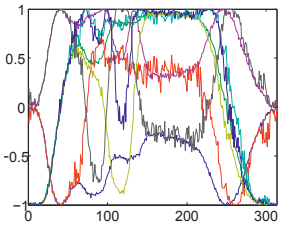
(a) Estimated target (b) Used target (c) ESN output after training with (b)

Figure 3.15: In experiment C the network was trained on one repetition of the YMCA movement with $\beta = 0.1$. The plots show (a) the estimated target, (b) the used target and (c) the ESN output after training with (b). All the plots are from epoch 5 of 10, where the used target is starting to look like the true target. Notice that the used target appears less noisy than the estimated target.

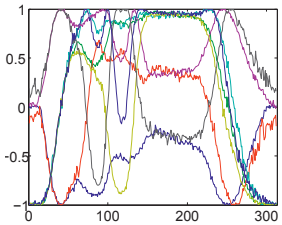


(a) Desired position

(b) Original, 1 rep.



(c) Original, 5 rep.



(d) New, $\beta = 0.1$

Figure 3.16: The plots illustrate the quality of the teacher forcing in experiment A, B and C. For each of these experiments the position sequences in epoch 5 of 10 are plotted as the 8 coordinate values at each time step for one repetition of the YMCA-movement.

3. Research Summary

DISCUSSION, CONCLUSION AND FUTURE WORK 4

This chapter summarizes the contributions, points out the limitations and suggests future work.

4.1 Discussion

The main objective of this thesis was to *study internal model based control schemes and how internal models can be learned and applied for artificial motor control*. This was motivated by the need for control architectures that can learn and that are able to control lightweight, highly complex robots with very many DOFs.

In particular, we focused on how internal models can be used to solve two control problems, namely delays in the sensory-motor-loop and bimanual coordination. We chose to represent the internal models as ESNs and to learn the inverse model with feedback-error-learning. Based on these decisions, four research questions were stated:

RQ1: *How can internal models be used to handle delays in the sensory-motor-loop?*
RQ2: *How can internal models be used to make one of the arms coordinate its movements relative to the movements of the other arm?*
RQ3: *What characterizes a training movement that makes an ESN generalize to most other movements?*
RQ4: *How can an ESN be trained with feedback-error-learning?*

Starting with the first research question; the idea of using forward models to handle delays in the sensory-motor loop is not new, but the forward model is then typically used together with a feedback controller, as in the Smith Predictor architecture. We added a forward model directly to the feedback-error-learning architecture, with the predicted state of the forward model as an additional input to the inverse model together with delayed sensed state and desired next state. To our knowledge this architecture has not been used before, and we believe it is worth further studying.

The work on bimanual coordination focused on what kind of patterns the ESN was able to learn and preserve as the velocity changed. In the control architecture the controller of

one arm received the state of both arms. Sharing the state, or position, is a common way to achieve coordinated movement. The novelty in our research lies in the training of an echo state network to learn the movement constraints. If we were to do the experiments again we would consider sharing motor commands instead of position. After all, the main difference between coordinating the movement of one limb with the other one compared with coordinating one's motion with someone else's, is the access to motor commands.

Most experiments in this thesis deal with the inverse kinematics problem of calculating motor commands that will move the limbs along a desired trajectory. The exceptions are the experiments on bimanual coordination. Most research on learning inverse kinematics focus on learning parts of the actuator space as they are needed. The general assumption seems to be that the robot needs to exhaustively explore the parts of the actuator space it is going to perform in. In our experiments we show that this is not necessarily true. For example, Skinny is able to execute movements using the full range of motion of some DOF even when he has only trained on a movement with minimal movement in that DOF.

Implementing the internal models with ESNs and training the inverse model with feedback-error-learning proved not to be as straight forward as originally thought, and a large part of this thesis was spent on RQ4. The result was a proposed new training method for ESNs that worked very well in the feedback-error-learning setup, and which might also be faster than the original training method on benchmark problems (thorough investigations are left to future work).

If we had started this work today, the progress might have been different, as a lot of research on ESNs has been conducted the latest years, and new recommendations for best practices have just been published [42]. The most important recommendations that we did not try ourselves are using ridge regression and regularization. It is not possible to know if it would have made any difference for our purpose, but it deserves to be tried.

Spending much time on the training of the internal models meant less time to study how to use them for motor control, i.e., RQ1 and RQ2. In experiments related to these questions we used a very simple simulator and a limited dataset, and we did not compare many different control architectures. Still, we were able to provide some interesting ideas and preliminary results.

4.2 Conclusion

The main contributions of this thesis can be summarized as follows:

- A promising control architecture, which appears to be able to handle delays in the sensory-motor-loop. The architecture consisting of a forward model coupled with an inverse model that is trained with feedback-error-learning. The novelty is that both predicted and delayed sensory feedback is used as input to the inverse model.

- Evidence that suggests exhaustive exploration of the parts of the actuator space used during performance is not necessary.
- Promising results with using ESNs to represent and learn internal models.
- A new training method for ESNs.

It can be concluded that this thesis has contributed to the initial objective of studying internal model based control schemes and how internal models can be learned and applied for artificial motor control.

4.3 Future Work

Improvements to the presented results could be made, and the following is a list of possible focus areas of future work:

- Run all experiments with other and more complex movement data, at least use the 3D dataset on all the experiments.
- Combine the different experiments, e.g., add delays in the sensory-motor loop during the bimanual experiments. In order to perform well-coordinated movements, the control systems of the two limbs would need direct or indirect access to each other's motor commands. For example, for this purpose it would be interesting to explore if each arm could have both an inverse model and a forward model as in the delay experiments. Instead of, or in addition to, having access to the sensed state of the other arm, the control system of one arm could have access to the prediction made by the other arm's forward model.
- Compare the use of ridge regression and regularization applied to the original training method with the proposed new training algorithm.
- Test the proposed new training algorithm on benchmark problems like generation of the "figure-eight" [84] or a chaotic attractor like the Mackey-Glass system [29] to see if it might be computationally faster than the original method.
- The new training method also makes it possible to gradually adapt the inverse model during performance. We turned the feedback controller off during testing, but in practical applications it should be available to handle potential changes in the simulator or the environment. As our method gradually adapts the output weights, short time perturbations will probably not lead to significant changes, but lasting changes in the plant should eventually change the inverse model. This is a hypothesis, which must be tested experimentally.

II

PUBLICATIONS

DANCING YMCA WITH DELAYED SENSORY FEEDBACK

A

Authors:
Rikke Amilde Løvlid and Pinar Öztürk

Abstract:
Lack of sensory feedback or delay in feedback has been shown to have detrimental effects on cognition and action in humans and in artifacts. Despite the adverse effects of delay, humans manage to generate smooth and timely movements. This has been explained by the existence of predictive models in the brain. In this paper we investigate the possible role of a predictive model that anticipates the consequences of the motor command to be issued to the actuator (e.g., arm). The paper presents two architectures, one with and the other without predictive components, and compares their performances in dancing to the song 'YMCA'. The architecture including the predictive model has been trained in three different ways to uncover the possible effects of the training method on the movement performance. The results confirm the role of prediction in the movement control.

Published in:
Proceedings of the Tenth IASTED International Conference on Artificial Intelligence and Applications, pages 144-150.

My main contributions to the paper:

- Programming the system and running the experiments
- Writing the paper

The co-author contributed to the following areas:

- Writing the paper, mostly the introduction.

DANCING YMCA WITH DELAYED SENSORY FEEDBACK

Rikke Amilde Løvlid
Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway
email: lovlid@idi.ntnu.no

Pinar Öztürk
Department of Computer and Information Science
Norwegian University of Science and Technology
Trodheim, Norway
email: pinar@idi.ntnu.no

ABSTRACT
Lack of sensory feedback or delay in feedback has been shown to have detrimental effects on cognition and action in humans and in artifacts. Despite the adverse effects of delay, humans manage to generate smooth and timely movements. This has been explained by the existence of predictive models in the brain. In this paper we investigate the possible role of a predictive model that anticipates the consequences of the motor command to be issued to the actuator (e.g. arm). The paper presents two architectures, one with and the other without predictive components, and compares their performances in dancing to the song 'YMCA'. The architecture including the predictive model has been trained in three different ways to uncover the possible effects of the training method on the movement performance. The results confirm the role of prediction in the movement control.

KEY WORDS
Neural Networks, Feedforward Control, Internal Models

1 Introduction

Studies on the human sensory-motor loop have shown that the time elapsed for the travel of neural signals from the brain to the muscles and the time the brain needs to process the sensory feedback it receives constitute a problem for the smoothness of voluntary movement. Humans perceive their own movement with a delay. For visual sensory feedback this delay can be as much as 200-250 ms, and for proprioception 110-150 ms [1]. Due to this delay, fast and coordinated movements cannot only be attributed to pure feedback [2, 1, 3].

The delay issue poses a problem in artifacts as well. Robots' electrical circuits do not suffer from the same conduction delays as the neurons in the human nervous system, but this does not necessarily mean that delay does not impair robots' movements. In order for a robot to behave properly in an environment, its movements should be timely and it should not overshoot the targets when, for example, reaching and lifting an object. For this, the robot needs to know its own current bodily state. However, depending on the complexity of the robot's perceptual capabilities, it might take time to process the sensory signals, e.g., to interpret the visual sensory input.

Effects of delayed sensory feedback have been of interest to researchers from various disciplines. Smith and Sussman [4] studied the cognitive effects of delay in tasks such as tracking, steering, handwriting, and head and body movements. Degradation in accuracy and timing was observed in all these tasks. In some tasks, peak disturbances was observed at specific delays while in others a deterioration of performance was detected in proportion to the introduced delay.

Another rather different situation where delay may be a problem is in remote manipulation systems, where the human operator is physically displaced from the machine under control. The lag in this situation is due to transmission delays in the communications channel, for example, as in controlling a space vehicle on the moon from the earth. MacKenzie and Ware [5] studied the delay effects on human performance in interactive systems, where delay could be attributed to properties of input devices, the software, and output devices. In their experiments, participants had to move a mouse from a starting point to a target location. A delay, changing between 25 ms and 225 ms, was introduced from moving the mouse to seeing the cursor move on the screen. Two important findings were the detection of the amount of delay that started to affect performance, which was - 75 ms, and the observation that more complex tasks lead to greater deterioration. Jay and Hubbold [6] tried to quantify the effects of latency on sensory feedback in distributed virtual environments and reported that haptic delay is less important than visual delay.

These were mostly examples of experiments where delay was intentionally introduced aiming to investigate its consequences. On the other hand, it was conceived that there is an internal delay in the sensory-motor loop of humans [1]. How come then, humans manage to produce, for example, smooth voluntary movements (e.g., lifting a cup in contrast to reflexive withdrawal of a hand from a hot plate) despite significant delays in the feedback? Neuroscience studies suggest that human brain hosts predictive models of the motor behavior of limbs in order to cope with the delay in sensory feedback [7, 8, 9, 10].

In this paper we investigate whether and when predictions may cancel the adverse effects of sensory feedback delay. In order to investigate whether predictive models similar to humans' may also render corresponding results in robots, we compared the performance of two architec-

tures (one with and another without a predictive model) with regard to the smoothness of voluntary movements. The results of the experiments show the positive influences of predictive models on the movements. In the experiments, the architectures were trained to imitate the movements of the dance to the song YMCA by The Village People. The second part of the work presented in the paper deals with the training methods applied to the architecture involving a predictive model. In the next section we provide a brief overview of the related work in the control theory, computational neuroscience, and robotics, along with a description of the terminology used in the rest of the paper. Section 3 introduces our hypotheses that constitute the motivation of this work, the training data used, the two architectures highlighting the possible role of predictive models, the experimental setup used in evaluating the hypotheses, and the neural network implementation of the controller and the predictive model. The results of the experiments are presented in section 4, and finally, in section 5 we wrap up with the discussion and conclusions.

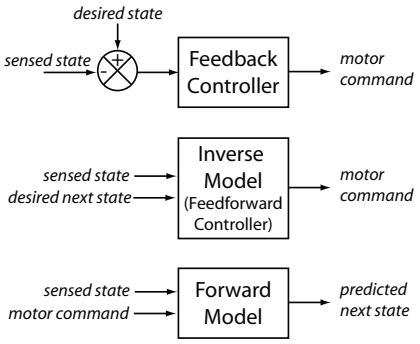


Figure 1. A feedback controller translates the difference between the desired and the sensed state into a motor command. An inverse model calculates the motor command based on the sensed state of the system and the desired next state. A forward model predicts the outcome of the motor command given the sensed current state.

2 Motor Control and Internal Models

Computational neuroscientists have adopted much of the terminology used in control theory and applied it to human voluntary motor control. This section gives a summary of this terminology and also explains how our experimental architectures fit in.

In control theory there are two basic types of control: *feedback-* and *feedforward control*. In feedback control the sensed state of the system is compared with the desired state of the system and adjustive motor commands are issued accordingly. In feedforward control the future desired state is monitored and a motor command is issued to drive the system towards this state. It can be said that a feedforward control system acts proactively, whereas a feedback control system is purely reactive. For example, a thermostat that turns on the heat when the temperature drops below the desired value and turns the heat off when the temperature is too high is a feedback controller. A corresponding feedforward controller could for instance turn on the heat whenever a window was opened, before the temperature drops to the critical point.

In some applications feedback control might be sufficient, but when the feedback loop is slow, to merely rely on the sensory feedback has proven to result in highly unstable control [11].

In order to implement feedforward control, one needs to know something about the *plant*, the system to be controlled. For example, in order to use the proactive controller regulating the temperature, one needs direct or indirect knowledge about the effect of opening a window on the room temperature. There are two types of feedforward control, *direct control* and *indirect control using internal models*.

Direct control means control without explicit knowledge of the behavior of the plant. The control policy, which maps the relevant internal and external states to actions, is essentially treated as a black box, and can be learned with reinforcement learning [12], or it can be implemented as rules. The equilibrium point hypothesis is a proposed direct control approach to voluntary motor control [13].

Most research today regarding voluntary motor control shows a tendency towards indirect feedforward control with the use of internal models. An internal model is a system that mimics the behavior of a natural process. In control theory, two types of internal models are emphasized, *forward models* and *inverse models*. A forward model predicts the outcome of an action (i.e., motor command). An inverse model represents the opposite process of calculating an action that will result in a particular outcome, the desired next state. Existence of internal models in the brain is widely accepted and there are many theories of how they are used and where they are located [3, 14, 10]. Figure 1 illustrates the input and outputs of a feedback controller, an inverse model and a forward model.

There are several ways to use forward and/or inverse models to accomplish indirect feedforward control. The straightforward approach is to use only an inverse model. Since the input-output function of the inverse model is ideally the inverse of the body's forward dynamics, an accurate inverse model will produce perfectly the desired trajectory it receives as input, as long as there are no external perturbations. To acquire such an accurate inverse model through learning is, however, problematic. Kawato [15] investigates different possibilities. Among these, we have adopted the feedback-error-learning scheme. In this scheme one uses a simple feedback controller together with the inverse model.

A second approach to feedforward control is to add a forward model to the simple feedback controller [15, 7].

The forward model would predict the next state based on the delayed sensed state and the motor command issued by the controller. The idea is that the prediction from the forward model will be available much earlier than the sensory feedback, and the controller can react faster by using the difference between this predicted state and the desired state, as opposed to the difference between the sensed state and the desired state. The feedback controller uses internal feedback provided by the forward model instead of external feedback. When the loop trough the forward model is fast, this architecture will in practice achieve the same result as an inverse model. Again, the question is how such a forward model can be acquired trough learning. The Smith Predictor [1] provides one solution to this problem of training the forward model.

As described, model based, indirect feedforward control can be acquired either with the use of an inverse model or by using a forward model. Wolpert et. al [3] and Smagt et. al [11] suggest a third option, namely to use both, a forward and an inverse model coupled together.

In this paper we investigate two of these three possibilities. The first is an inverse model coupled with a feedback controller for feedback-error-learning and online correction. In the second architecture a forward control is added to the first architecture. We do not look at the possibility of using only a forward model together with a feedback controller without the inverse model.

3 Dancing YMCA with Delayed Sensory Feedback

This section explains the two architectures we used in order to study the role of predictive models in resolving negative effects of delay, the experimental setup used, details of the implementation of the inverse and forward models, and training of the whole system.

3.1 Hypotheses

An inverse model could, in theory, tackle the delay on its own, as explained in section 2, but it requires a very accurate inverse model, which is often hard to acquire. In this paper, we investigate whether and when a forward model (i.e., a predictive model) needs to be used together with the inverse model to reduce the adverse consequences of delay when the inverse model is imperfect. Our aim can be conveyed through the following two hypotheses:

Hypothesis 1: *The performance of the system will be better with a predictive model than without when delay is significant.*

Hypothesis 2: *Pre-training the predictive model will increase the performance.*

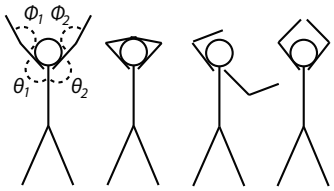


Figure 2. The Movement is described with four angles; the two shoulder abduction angles $\theta 1$ and $\theta 2$ and the elbow extension angles $\phi 1$ and $\phi 2$.

3.2 The Movement Data

In the implementation of the experiments, we trained the architecture to imitate the dance to the song YMCA by The Village People (se figure 2). Movement data was gathered with a Pro Reflex 3D motion tracking systems by Axel Tidemann [16].

The movement of each arm was described in two degrees of freedom, the angle between the arm and the body, i.e., the abduction angle θ, and the angle between the under- and upper arm, i.e., the extension in the elbow ϕ. Hence, the simulated robot was described by 4 degrees of freedom.

The sampling frequency used in the experiments was 50 samples/sec, which means that each time step is 0.02 sec. The whole movement takes 6.26 sec, i.e., 313 time steps. The YMCA movement, hence, is represented as a sequence of 313 states. This sequence was used as the training data where state t represents the desired state at time step t. The goal of the control system is to produce motor commands that generate a movement where the state generated in each time step is as close as possible to the corresponding state in the desired sequence of states.

3.3 The Architectures

The control architectures that underlie the experiments are shown in figure 3. The control system consists of a feedback controller, an inverse model and possibly a forward model. At each time step t the control system receives as input the *desired next state* (i.e. the joint angles at time step *t+1* in the training sequence) and the *sensed state* representing the actual state of the plant a given number of time steps earlier (specified by "Delay" in figure 3), and outputs a *motor command*.

The inverse model receives as input the desired next state, the sensed state (i.e., delayed actual state) and, when the forward model is included, the predicted current state, and produces a motor command. The feedback controller translates the difference between the desired state and the actual state (i.e. state error) to a motor command that is used to adjust the motor command produced by the inverse model. This means that the error done in time step t is used to adjust the motor command in the time step *t+1*.

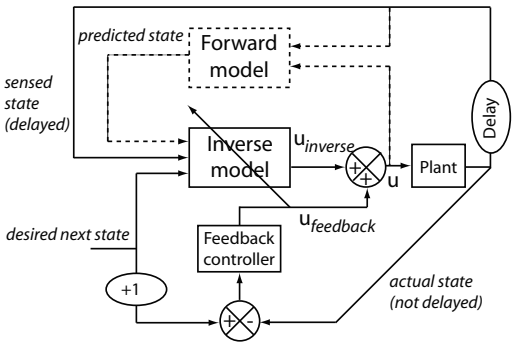


Figure 3. Two control architectures, one has only an inverse model while in the other, the inverse model is coupled with a forward model. The dotted loop including the forward model is only enabled in the experiments where the forward model is involved. Delays are shown with ellipses (+1 means delayed with one time step), an arrow across a model represents training, and **u** is the motor command.

The total motor command is sent to the plant, i.e., the robot simulator.

Note that the state information from the plant to the feedback controller is not delayed, as is the case for the state information from the plant to the internal models. It has been suggested that while visual feedback is vital for feedforward control, feedback control relies on proprioceptive feedback [17]. Although we did not make an explicit distinction between the different modalities of feedback, the idea was that the feedback controller relies on a proprioceptive-like feedback which can be assumed not to be delayed when compared with the feedback (e.g. visual) used as input to the inverse and forward models.

When included, a forward model was trained to predict the next state based on the motor command and the sensed, delayed state.

This architecture differs from other proposed architectures involving a pair of forward and inverse models in using both delayed sensory feedback and the prediction from the forward model as input to the inverse model. The alternatives would be to use only the prediction as input to the inverse model or to integrate the prediction and the delayed sensory feedback prior to feeding it to the inverse model [18, 19, 20].

A detailed description of the implementation and training of the different modules is given in section 3.5.

Normal-distributed noise with standard deviation 0.01 was added to the sensory signal from the plant. The system was implemented in MatLab, including a simple stick-man-simulator used as plant.

3.4 Experiments

In section 3.1 two hypotheses were stated, (1) *the performance of the system will be better with predictive models than without when delay is increased*, and (2) *pre-training the predictive model will increase the performance.*

To test the first hypothesis the two architectures were compared. The first consisting only of an inverse model and the second consisting of an inverse model coupled with a forward model. A feedback controller for online correction of the motor commands is included in both architectures to ensure stability. The architectures are illustrated in figure 3.

In order to investigate the second hypothesis, training of the models in the architecture with a forward model was examined in three modi: *parallel*, *serial* and *hybrid*. In the parallel training modus the forward and the inverse models were trained in parallel. In the serial and the hybrid modus the forward model was pre-trained. That is, the forward model was trained before the whole system started to learn the desired movement. In the hybrid modus, different from the serial modus, the forward model continued it's training while the inverse model was being trained.

To pre-train the forward model we used the desired movement; the forward model was trained to predict the next state in the sequence of desired states given the correct motor command and the delayed and noisy sensory signal.

In summary, in order to study the role of the forward model and the effects of the training method, we conducted four sets of experiments:

Experiment 1:	*Only Inverse.*
Experiment 2:	*Inverse and Forward, training modus parallel.*
Experiment 3:	*Inverse and Forward, training modus serial.*
Experiment 4:	*Inverse and Forward, training modus hybrid.*

The result of experiment 1 has been compared with the results of the experiments 2, 3 and 4 to test the first hypothesis. In the investigation of hypothesis 2, results of the three last experiments are compared.

The experiments were all run on the same task, namely imitating the YMCA dance. The delay was varied between 0 and 10 time steps. All experiments were run the same number of epochs to make the comparison easy.

3.5 Forward- and Inverse Models

Both the inverse and the forward models were implemented as echo state networks [21] with 1000 and 100 internal nodes in the hidden layer respectively. Both networks had spectral radius $\alpha = 0.1$ (determining the length of the memory with increasing $\alpha \in [0, 1]$) and noise level $v = 0.2$ (effectively adding 10% noise to the internal state of the network). The inputs and outputs where scaled to be in the range [-1,1].

The inverse model has 8 input nodes when the forward model is not included, 4 representing the sensed state and 4 representing the desired next state. When the forward model is included, the inverse model has 4 additional inputs representing the predicted state. The output layer of the inverse model has 4 nodes, where each node corresponds to the motor command for one joint angle. The forward model has 8 input nodes, the sensed state and the motor commands for each joint angle, and 4 output nodes representing the predicted state. There were no connections from the output layer to the input- or hidden layer in either of the models.

Training of the inverse model was done with the help of a feedback controller. In this experiment the arm model is very simple, which makes it easy to compute the motor command corresponding to the measured difference between the desired and the actual states analytically. This motor command ($u_{feedback}$) has a double role: it is is used to train the inverse model, and the feedback controller adds it to the final motor command to pull the system in the right direction. The $u_{feedback}$ gain K was 0.65, and the average ratio $u_{feedback}/u$ was monitored together with the performance error.

The input-, internal- and output weights of the two networks were initially generated randomly and the networks were then trained with linear regression as described by Jaeger [21].

4 Results

All experiments were run 10 times. For each experiment, the mean and standard deviation for the performance error, prediction error and ratio online correction, $u_{feedback}/u$ are presented.

Performance error is the mean difference between the desired state and the actual state produced by the system in each time step after the last epoch of training.

Online correction ratio, $u_{feedback}/u$, is the feedback controller's contribution to the total motor command in percent. Since the feedback controller adjusts the motor command based on the performance error in the previous time step, the higher this ratio, the less the inverse model has learned. Therefore, this ratio constitutes, in addition to the performance error, an important criteria in the evaluation of the system.

Prediction error is the mean error done by the forward model during the last epoch, and is used when comparing the systems performance in the different training modi when the forward model is included in the architecture.

4.1 With and without Forward Model

This section discusses the results from the experiments that were conducted in order to see whether our first hypothesis could be supported (i.e., the performance of the system will be better with predictive models than without when delay is increased).

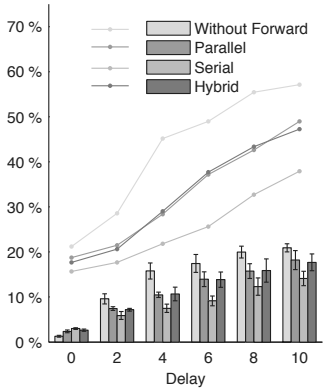


Figure 4. The figure shows a comparison of performance errors and online correction ratio with only inverse model and with forward model and inverse model trained in the three modi, *parallel*, *serial* and *hybrid*. Bars show the mean performance error with standard deviation for each of the three methods of training. The curves show the mean feedback ratio.

The mean performance error and the mean ratio of online correction for all the four sets of experiments are illustrated in figure 4. With regard to hypothesis 1, we compare the results of the experiment without a forward model and the three experiments with a forward model. Results from the training of the forward models in the three different modi will be elaborated in the next section.

Without a forward model the performance error increases rapidly to 10-20% with increasing delay, but equally important is the increase in online correction. With only four time steps delay almost half of the motor command is produced by the feedback controller. This means that the inverse model has clearly problems learning the movement.

The performances of all the three experiments with a forward model look better than the one without, for delay ≥ 2. Both the mean performance error and the ratio online correction are significantly lower when a forward model is included, regardless of the training modi.

The architecture with the forward model trained in serial modus shows the best performance of all the setups when there is a delay. In fact, when delay is ≥ 4 the best performance measured with the system without forward model is always worse than the worst performance measured with the system with forward model trained in serial modus. This is illustrated in figure 5.

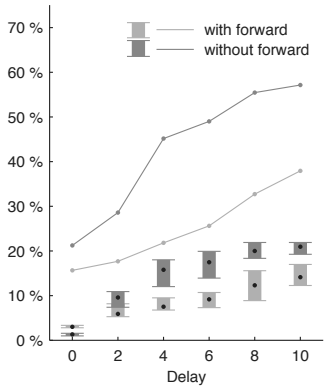


Figure 5. The figure shows a comparison of the performance and the online correction ratio with and without a forward model. Serial training was used when a forward model was involved. The interval shows the min and max performance error in the last epoch over all 10 runs. For delay ≥ 4 the the performance error for the worst run with forward is better than the performance error for the best run without.

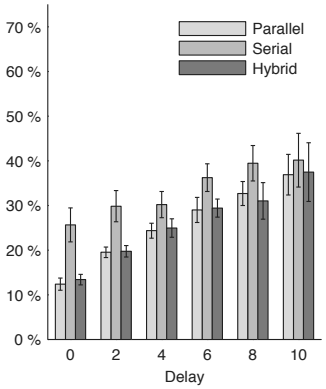


Figure 6. The figure shows a comparison of the prediction errors when training the forward and the inverse models in *parallel*, *serial*, and *hybrid* modi. Bars show the mean prediction errors with standard deviation for each of the three training modi.

4.2 The Three Training Modi with a Forward Model

As mentioned in the preceding section and illustrated in figure 4, the serial training modus shows significantly better performance than the other training modi. We found no significant difference in performance between parallel an hybrid training modi.

Serial training yields better performance in spite of significantly higher prediction error than in the other two modi, as can be seen in figure 5. Why this might be happening will be discussed in section 5.

The prediction error in the last run of pre-training was approximately 8% for 0 delay and 9% for 10 delay.

5 Discussion and Conclusion

The results of the experiments presented in section 4 lend support to hypothesis 1, i.e., a forward model that predicts the outcome of an action provides useful information to the inverse model when sensory feedback is delayed.

Hypothesis 2 falls short to capture the whole story about the training method. We expected that pre-training would provide a forward model with a higher predictive capability. The results do not align with our expectations. During pre-training the prediction error was very low, but good prediction capabilities on the training sequence does not automatically lead to good prediction during training of the inverse model. This is clear from the results showing higher prediction errors in serial modus during the training of the inverse model, and indicates that the forward model generalizes poorly. A longer sequence of a movement as the training data may improve the generalization of the forward model.

Despite the higher prediction error during training of the inverse model, training in serial modus still yields the best performance. We hypothesize that this might be because the forward model is pre-trained to predict the correct movement and therefore imposes some kind of bias on the inverse model. This will be tested in future work by training the forward model with babbling instead of the desired movement before training the inverse model.

Using the correct movement to train the forward model assumes that these motor commands are known. This might make the training of the inverse model seem pointless, but the reason to store a movement in a neural network in this way might make the control system generalize to other movements. This will be investigated in further studies.

We could not see any difference between pre-training and not of the forward model when the forward and inverse models where continued to be trained in parallel, which means that what was pre-learned is lost during the concurrent training. Had another learning algorithm been used, for example back-propagation with decaying learning rate the result might have been different.

Further studies should include more complex movements and other learning algorithms like Backpropagation-Decorrelation [22]. It would also be interesting to investigate how the accuracy of the inverse model effects its ability to handle delay, different ways to train the inverse model and how accurate the forward model must be in order to be of any help.

References

[1] R. Miall, D. Weir, D. M. Wolpert, & J. Stein, Is cerebellum a smith predictor? *Journal of Motor Behavior*, *25*(3), 1993, 203–216.

[2] M. Kawato, Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, *9*(6), 1999, 718–727.

[3] D. M. Wolpert, R. C. Miall, & M. Kawato, Internal models in the cerebellum. *Trends in Cognitive Sciences*, *2*(9), 1998.

[4] K. U. Smith & H. M. Sussman, Delayed feedback in steering during learning and transfer of learning, *Journal of Applied Psychology*, *54*, 1970, 334–342.

[5] I. S. MacKenzie & C. Ware, Lag as a determinant of human performance in interactive systems. In *Proc. of the INTERACT '93 and CHI '93 conference on Human factors in computing systems*, New York, 1993.

[6] C. Jay & R. Hubbold, Quantifying the effects of latency on sensory feedback in distributed virtual environments, In *Proc. of Virtual Images Seminar* 2006, 9–16.

[7] B. Mehta & S. Schaal, Forward models in visuomotor control, *Journal of Neuropysiology*, *88*(2), 2002, 942–953.

[8] , A. G. Witney, S. J. Goodbody, & D. M. Wolpert. Predictive motor learning of temporal delays, *Journal of neurophysiology*, *82*(5), 1999, 2039–2048.

[9] R. Miall & D. M. Wolpert, Forward models for physiological motor control, *Neural Networks*, *9*(8), 1996, 1265–1279.

[10] P. R. Davidson & D. M. Wolpert, Widespread access to predictive models in the motor system a short review, *Journal of Neural Engineering*, *2*(3), 2005, 313–319.

[11] P. van der Smagt & G. Hirzinger, The cerebellum as computed torque model, In R. Howlett & L. Jain, eds., *Fourth International Conference on Knowledge-Based Ingelligent Engineering Systems & Applied Technologies*, 2000.

[12] M. L. L. Leslie Pack Kaelbling & A. W. Moore, Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, *4*, 1996, 237–285.

[13] R. Shadmehr. Equilibrium point hypothesis, In M. A. Arbib, ed., *The Handbook of Brain Theory and Neural Networks, second edition*, 409–412 (The MIT Press, 2002).

[14] R. Shadmehr & J. W. Krakauer, A computational neuroanatomy for motor control, *Experimental Brain Research*, *185*(3), 2008, 359–381.

[15] M. Kawato & H. Gomi. The cerebellum and vor/okr learning models. *TINS*, *15*(11), 1992.

[16] A. Tidemann & P. Öztürk, Self-organizing multiple models for imitation: Teaching a robot to dance the YMCA, In *IEA/AIE*, vol. 4570 of *LNCS* (Springer, 2007), 291–302.

[17] E. R. Kandel, J. H. Schwartz, & T. M. Jessel, *Principles of Neural Science*, chap. 33, 656–657 (McGraw-Hill, 2000).

[18] D. M. Wolpert & Z. Ghahramani, Computational principles of movement neuroscience, *Nature neuroscience*, *3*, 2000, 1212–1217.

[19] D. M. Wolpert, S. Goodbody, & M. Husain, Maintaining internal representations: the role of the human superior parietal lobe, *Nature Neuroscience*, *1*(6), 1998, 529–533.

[20] R. Grush, The emaluation theory of representation: Motor control, imagery, and perception, *Behavioral and Brain Sciences*, *27*, 2004, 377–442.

[21] H. Jaeger, A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach, Techreport, Fraunhofer Institute for AIS, 2008.

[22] J. J. Steil, Backpropagation-decorrelation: online recurrent learning with o(n) complexity, In *Proc. IJCNN*, 2004.

LEARNING BIMANUAL COORDINATION PATTERNS FOR RHYTHMIC MOVEMENTS

B

Authors:
Rikke Amilde Løvlid and Pinar Öztürk

Abstract: Coordinated bimanual movements form the basis for many everyday motor skills. In human bimanual coordination there are several basic principles or default coordination modes, such as the preference for in-phase or anti-phase movements. The objective of our work is to make robots learn bimanual coordination in a way that they can produce variations of the learned movements without further training. In this paper we study an artificial system that learns bimanual coordination patterns with various phase differences, frequency ratios and amplitudes. The results allow us to speculate that when the relationship between the two arms is easy to represent, the system is able to preserve this relationship when the speed of the movement changes.

Published in:
Volum 6354 of *Lecture Notes in Computer Science*, pages 143-148.

My main contributions to the paper:

- Programming the system and running the experiments
- Writing the paper

The co-author contributed to the following areas:

- Writing the paper, mostly the introduction.

Learning Bimanual Coordination Patterns for Rhythmic Movements

Rikke Amilde Løvlid and Pinar Öztürk

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract. Coordinated bimanual movements form the basis for many everyday motor skills. In human bimanual coordination there are several basic principles or default coordination modes, such as the preference for in-phase or anti-phase movements. The objective of our work is to make robots learn bimanual coordination in a way that they can produce variations of the learned movements without further training. In this paper we study an artificial system that learns bimanual coordination patterns with various phase differences, frequency ratios and amplitudes. The results allow us to speculate that when the relationship between the two arms is easy to represent, the system is able to preserve this relationship when the speed of the movement changes.

1 Introduction

Humans use both hands in most of their daily tasks. Bimanual movements have, therefore, been rigorously studied in various research disciplines. There are mainly two dominant frameworks for the theory of interlimb coordination, the *dynamic pattern theory* [1, 2] and *crosstalk theory* [3, 4]. Dynamic pattern theory aims at a mathematical formalization of the coordination principles, modeling rhythmic movements as a system of coupled nonlinear oscillators [4]. The main idea in neural crosstalk theory is that interactions occur between command streams within a highly linked neural medium. These will give rise to patterns of mutual interference between concurrent limb motions at different stages of movement planning and organization [3].

In contrast to plentiful studies in psychology and neuroscience, robotic studies of bimanual coordination are surprisingly elusive. If robots are aimed to truly assist humans, they should be able to learn mastering movements that require bimanual coordination.

Our work is inspired by the hypothesis that the crosstalk-based bimanual coordination requires a peculiar connection between the motor system of the two limbs to be coordinated [5]. This paper presents an architecture that models the cross talk between two arms where the position of one arm is communicated to and controls the movement of the other arm, hence bimanual coordination. Our aim is not to investigate how the neural crosstalk happens in the nature but to develop an architecture that mimics the bimanual capabilities of humans.

The paper proposes a method to study the characteristics of the movements that the cross talk architecture can afford (i.e., learn and produce). Three properties are investigated to characterize and analyze the affordance of the architecture: *relative phase*, *frequency*, and *amplitude*. Then, the speed changes have been used as the evaluation criterion to evaluate the stability of the architecture with respect to the identified movement characteristics.

Our results indicate that the architecture acquires various movements through learning how to represent the interaction between the limbs, rather than each controller learning its movement in isolation.

2 The Method

The following three properties pertinent to a movement characterize the class of movement that the architecture is able to learn and produce.

Relative phase is the extent of phase lag between two joints at any point in time. Generally humans show a basic tendency towards perfectly synchronized, in-phase (phase difference $\Phi = 0°$), or anti-phase movements ($\Phi = 180°$) [6].

The *frequency* is the number of occurrences of a rhythmic, repeating movement per unit time. When doing rhythmic movements with both arms, humans have a tendency towards rhythms where the frequency of one arm is an integer multiple of the frequency of the other (e.g. 1:1, where both arms move together, 1:2, where one arm completes two cycles at the same amount of time as the other completes one, 1:3, etc.), as opposed to polyrhythms (e.g. 2:3 or 3:5) [4].

One limb moves with a larger *amplitude* than the other if it moves further than the other in a given time interval. For humans the amplitudes of the two arms have a tendency to become similar [4].

We elaborate the characteristics of movements in terms of these properties. Once a movement with one of these characteristics is learned, we test whether the cross talk architecture can preserve this characteristic with a new speed.

3 Architecture and Implementation

The bimanual coordination architecture relies on a simple crosstalk where the control system controls only the movement of the lagging arm; the movement of the leading arm is forced. The input to the system is the current states (the arms' joint angles), while the output is the motor command to the lagging arm.

The basic movement used in the experiments was simply raising and lowering the arms, with no movement in the elbow joint. The movement of each arm was described in two degrees of freedom, the angle between the arm and the body and the angle between the under- and upper arm. Hence, the simulated robot was described by 4 degrees of freedom.

The system architecture is shown in figure 1 and consists mainly of an inverse model. Generally an inverse model calculates an action that will bring the system to the desired next state. In the current setting, the desired next state is

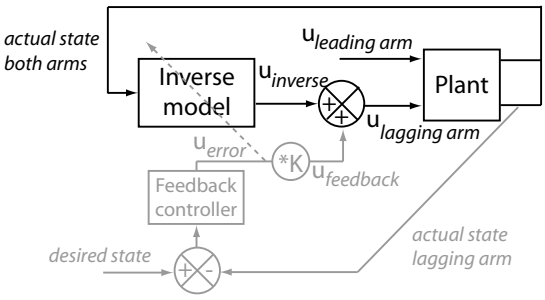


Fig. 1. The control architecture. An arrow represents training, and u is a motor command.

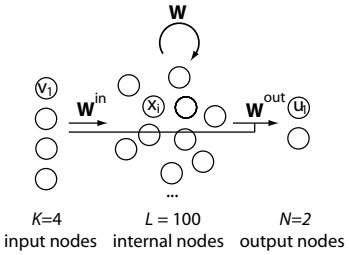


Fig. 2. The Inverse model as an Echo State Network.

implicit in the state of the leading arm. The inverse model is acquired through *feedback-error-learning* as suggested by Kawato, where a simple, analytical feedback controller is used together with the inverse model [7].

The inverse model was implemented as an *echo state network* (ESN) [8] as illustrated in figure 2. The input to the inverse model, v, is the current state from the plant (the robot simulator);

$$\mathbf{v}^{t+1} = plant(\mathbf{u}^t_{inverse} + K\mathbf{u}^t_{feedback}, \mathbf{u}^t_{leading\ arm}). \tag{1}$$

The feedback gain K, which decides how much the feedback controller is able to influence the final motor command, was linearly decreased from 1 to 0 during training. The activation of the internal nodes was updated according to

$$\mathbf{x}^{t+1} = f(\mathbf{W}^{in}\mathbf{v}^{t+1} + \mathbf{W}\mathbf{x}^t), \tag{2}$$

were f i the activation function. The motor command calculated by the inverse model is given by

$$\mathbf{u}^{t+1}_{inverse} = f^{out}(\mathbf{W}^{out}(\mathbf{v}^{t+1}, \mathbf{x}^{t+1})). \tag{3}$$

The training procedure was organized in epochs and cycles, where one cycle is one full temporal presentation of the training motion. Each epoch consisted of seven cycles. First, the network was re-initialized by setting the internal states of the network to zero, and one cycle was ran without updating the weights. Subsequently, the training sequence was presented five times with enabled learning. The output connections were then adapted after each complete cycle. A final cycle was used to estimate the performance error on the training sequence while learning was disabled.

During the training cycles, the state of each node was stored in a state collection matrix, $\mathbf{M}$, and $(f^{out})^{-1}(\mathbf{u}_{target})$ was collected row-wise into a target collection matrix, $\mathbf{T}$. Equation 3 can now be written

$$\mathbf{M}(\mathbf{W}^{out})^T = \mathbf{T}, \tag{4}$$

and solved with regard to $\mathbf{W}^{out}$ with the use of the Moore-Penrose pseudoinverse:

$$(\mathbf{W}^{out})^T = \mathbf{M}^+\mathbf{T}. \tag{5}$$

Note that it is not straight forward to decide how to compute the target motor command, $\mathbf{u}_{target}$, used in the target collection matrix. The desired motor command is only known indirectly trough the state of the leading arm. Generally, several motor commands may result in the same position of the limbs, and one does not want to bias the controller into choosing one specific solution. Because we had batch learning, the best guess could be calculated by using the u_error provided in the next time step, $u^t_{best\ guess} = u^t + u^{t+1}_{error}$ (e^t_{error} reflects the error done at time step $t-1$). However, using $u_{best\ guess}$ as the target did not lead to the best solution. In previous work we have shown that using something in between the best guess and the actual output of the inverse model in the previous cycle yields better results [9]. This result was confirmed in the current experiments. The target state for time step t in epoch $k+1$ was then given by:

$$u^{t,k+1}_{target} = \beta u^t_{best\ guess} + (1-\beta)u^{t,k}_{inverse} \tag{6}$$

In the experiments we used $\beta = 0.01$.

After the inverse model was trained on one movement, we wanted to test how the changes in the movement of the leading arm would affect the movement of the lagging arm. This was done by changing the speed of the leading arm and run the network for one additional epoch with only two cycles, the initialization cycle and the evaluation cycle. The feedback gain K was 0 during testing.

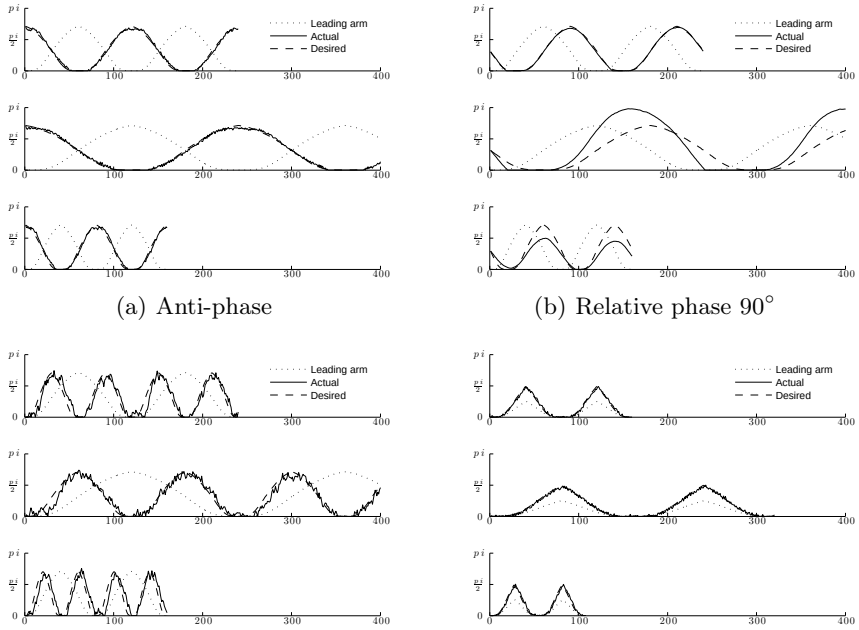
4 Experiments and Results

In the experiments the control system was trained to learn a coordinated movement. The stability of different coordination patterns was compared by testing the system's ability to replicate the learned pattern in different speeds.

Relative Phase: In this experiment the arms were trained to move in a specific phase relationship. It could be in-phase ($\Phi = 0°$), i.e., both arms are raised and lowered synchronously, or anti-phase ($\Phi = 180°$), i.e., one arm is lowered as the other is raised, or anything in between. During the training, the control system learned to perform one specific phase relationship for one motion in one speed. During testing, the speed of the leading arm was changed.

The control system had no problem generalizing to new speeds when it was trained (and tested) with in-phase or anti-phase movements, as illustrated in figure 3(a).

The results of testing the system in higher or slower speeds than it was trained for when the two arms are in 90° phase difference is shown in figure 3(b). The control system is trying to coordinate its movement according to the movement of the leading arm, instead of sticking to the absolute timing it was trained on, but it does not accomplish this as well as it did in the in- and anti-phase modi. The control system has a hard time matching the speed of the leading arm. When the leading arm moves faster, the lagging arm also moves faster, but not as much as the leading arm. Equivalently, when the speed of the leading arm is decreased, the lagging arm also moves slower, but not slow enough. Instead of changing direction at the trained state, the leading arm overshoots or undershoots the



(a) Anti-phase (b) Relative phase 90°

(c) Frequency ratio 1:2 (d) Different amplitude

Fig. 3. Results when the trained control system was tested on the same movement in the original speed (top), half the original speed (middle) and on 50% increase of the original speed (bottom). The shoulder angle of the leading arm and the desired and actual shoulder angle of the lagging arm at each time step is plotted. The movement of the leading arm is shown together with the desired and actual movement of the controlled lagging arm. (a) The generalization to different speeds is close to perfect for the anti-phase movement. (b) The phase difference of 90° seems harder to preserve when speed is changed. (c) The system was trained to perform the basic movement with frequency ratio 1:2. The movement of the lagging arm is not smooth, but the timing is just as good in the testing case as in the training case. (d) The lagging arm moves twice as far within the same time interval as the leading arm. The control system has no problem generalizing.

target when moving too fast or too slow respectively. As a consequence, the timing of the change in direction is closer to the desired than what it would have been if the change in direction had been timed solely according to the state of the lagging arm.

Frequency: The objective in this experiment was to study the control system's capability to perform a different rhythm with each arm. The control system trained both with simple rhythms, with frequency ratio 1:2 (leading arm moves ones while lagging moves twice) and 1:3, and more complex polyrhythms with frequency ratio 2:3 and 3:5.

This proved to be quite difficult. Training with frequency ratio 2:3 failed, and as illustrated in figure 3(c) (top), the performance is far from smooth for frequency 1:2. However, the timing for frequency 1:2 is satisfactory, and the results when testing the network trained on novel speeds is not worse than when tested on the original speed, as illustrated in figure 3(c) (middle and bottom).

Amplitude: In this experiment the arms were trained to move with different amplitude with respect to each other, one arm making a larger movement than the other, i.e. the motor commands of one of the arms must be twice as large as the other's. This appeared to be an easy task to learn, and the control system generalized perfectly to novel speeds as illustrated in figure 3(d).

5 Discussion and Conclusion

The aim of the work was to develop an artificial system that could learn and generalize different bimanual coordination patterns related to relative phase, frequency ratio and amplitude in the proposed architecture. As it is with humans, in-phase and anti-phase movements was more stable than the movements with other phase relationships. As for frequency ratio, polyrhythms like 2:3, appeared to be impossible to learn, whereas simpler rhythms like 1:2 and 1:3 seemed relatively stable once learned. The control system had no problem generalizing when it was trained with different amplitudes on the two arms. These results allow us to speculate that when the relationship between the two movement components is easy to represent, the system is able to preserve this relationship when the speed of the movement changes. In the continuation of this work we will try to more formally define the characteristics of the movement that are easier to coordinate, and what is easy to represent.

References

1. Kelso, J.A.S.: Dynamic patterns: The self-organization of brain and behavior (1995)
2. Kelso, J.A.S., de Guzman, G.C., Reveley, C., Tognoli, E.: Virtual partner interaction (vpi): exploring novel behaviors via coordination dynamics. PLoS One **4**(6) (2009)
3. Banerjee, A., Jirsa, V.K.: How do neural connectivity and time delays influence bimanual coordination? Biological Cybernetics **96**(2) (2007) 265–278
4. Swinnen, S.P.: Intermanual coordination: From behavioural principles to neural-network interactions. Nature **3** (2002) 348–359
5. Diedrichsen, J., Shadmehr, R., Ivry, R.B.: The coordination of movement: optimal feedback control and beyond. Trends in Cognitive Sciences (in press)
6. Peper, C.E., Beek, P.J.: Modeling rhythmic interlimb coordination: The roles of movement amplitude and time delays. Human Movement Science **18** (1999) 263–280
7. Kawato, M., Gomi, H.: The cerebellum and vor/okr learning models. TINS **15**(11) (1992)
8. Jaeger, H.: A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach. Techreport, Fraunhofer Institute for AIS (2008)
9. Løvlid, R.A.: Introducing learning rate analogy to the training of echo state networks. In: First Norwegian Artificial Intelligence Symposium. (2009)

LEARNING MOTOR CONTROL BY DANCING YMCA

C

Author:
Rikke Amilde Løvlid

Abstract:
To be able to generate desired movements a robot needs to learn which motor commands move the limb from one position to another. We argue that learning by imitation might be an efficient way to acquire such a function, and investigate favorable properties of the movement used during training in order to maximize the control system's generalization capabilities. Our control system was trained to imitate one particular movement and then tested to see if it can imitate other movements without further training.

Main Results:

IFIP Advances in Information and Communication Technology, (2010) 79-88

Learning Motor Control by Dancing YMCA

Rikke Amilde Løvlid

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract. To be able to generate desired movements a robot needs to learn which motor commands move the limb from one position to another. We argue that learning by imitation might be an efficient way to acquire such a function, and investigate favorable properties of the movement used during training in order to maximize the control system's generalization capabilities. Our control system was trained to imitate one particular movement and then tested to see if it can imitate other movements without further training.

1 Introduction

Humanoid robots assisting humans can become widespread only if they are easy to program. This might be achieved trough learning by imitation, where a human movement is recorded and the robot is trained to reproduce it. However, to make learning by imitation efficient, good generalization capabilities are crucial. One simply cannot demonstrate every single movement that the robot is supposed to make.

How we want the agent to generalize depends on what we want the agent to do. When watching the demonstrator move, the robot can either learn to mimic the motion of the demonstrator or learn how the demonstrator acts in many situations, that is, extracting the intention of the movement. Mimicking the exact movement trajectory might be important when learning a dance movement, but this is less important when washing the dishes. Belardinell et al. taught a robot to extract salient features from a scene by imitating the gaze shifts of a human demonstrator [1]. Wood and Bryson used observations of an expert playing a computer game to make the agent learn what contexts are relevant to selecting appropriate actions, what sort of actions are likely to solve a particular problem, and which actions are appropriate in which contexts [2].

Learning by imitation is, in a sense, something in between pre programming the agent's control policy (i.e., the function that decides which action to choose in every situation), and letting the agent figure it out on its own through trial and error. According to a hypothesis in developmental psychology, learning to control ones own motor apparatus may be based on so called motor babbling, i.e., random exploration of joint angles [3, 4]. Other findings suggest that children use more goal directed movements [5].

We argue that imitation can be used in an efficient way in learning to master the motor apparatus. In this paper we investigate the features of the training movement required to make the suggested control system generalize to new movements, and illustrate how imitation can be used to make the agent train on movements that are most valuable in terms of future generalization capabilities.

2 Feedforward Control

The goal of the work presented here is to make the agent capable of moving its limbs to the positions it desires, that is, we want the agent to learn *feedforward control*. In feedforward control the future desired state is monitored and a motor command is issued to drive the system towards this state. We could call this proactive motor control. The purely reactive alternative is *feedback control*, where the state of the system is compared with the desired state of the system and adjustive motor commands are issued accordingly. Often both feedforward- and feedback control is needed. In our experiments we have used a feedback controller to train the feedforward controller.

We consider feedforward control as a modular process where the control policy, i.e., the function that maps the current state and the future goal to a motor command, is decomposed into a *planning stage* and an *execution stage*. The planning stage generates a desired trajectory. This can be realized by generating the whole desired sequence in advance, or through a next state planner. In the presented work planning is done by the demonstrator, and our focus is on the execution stage.

2.1 Realization of Feedforward Control with Internal Models

There are several ways to realize the execution stage in feedforward control, but most research regarding voluntary motor control shows a tendency towards the use of *internal models*. An internal model is a system that mimics the behavior of a natural process. In control theory, two types of internal models are emphasized, *forward models* and *inverse models*. A forward model predicts the outcome of an action (i.e., motor command). An inverse model represents the opposite process of calculating an action that will result in a particular outcome, the desired next state. Existence of internal models in the brain is widely accepted and there are many theories of how they are used and where they are located [6–8].

Forward models, inverse models and feedback controllers can be combined in different ways to calculate the desired motor command [9, 10, 6, 11]. The straightforward approach is to use only an inverse model. Since the input-output function of the inverse model is ideally the inverse of the body's forward dynamics, an accurate inverse model will perfectly produce the desired trajectory it receives as input. To acquire such an accurate inverse model through learning is, however, problematic. Kawato investigated different possibilities [9]. Among these, we have adopted the *feedback-error-learning* scheme, where a simple feedback controller is used together with the inverse model. The details are explained in section 3.2.

2.2 Implementation of the Inverse Model

In our control system, the inverse model was implemented as an *echo state network* (ESN) [12]. The basic idea with ESNs is to transform the low dimensional temporal input into a higher dimensional *echo state* by using a large, recurrent neural network (RNN), and then train the output connection weights to make the system output the desired information.

Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks, and also simpler feedforward networks.

A typical task can be described by a set of input and desired output pairs, $[(i_1, o_1), (i_2, o_2), ..., (i_T, o_T)]$ and the solution is a trained ESN whose output y_t approximates the teacher output o_t, when the ESN is driven by the training input i_t.

Initially, a random RNN with the Echo State property is generated. Using the initial weight matrixes, the network is driven by the provided input sequence, $[i_1, i_2, ...i_n]$, where n is the number of time steps. Teacher forcing is used, meaning o_t is used instead of y_t when computing the state of the network at $t + 1$. The state of each node at each time step is stored in a state collection matrix, $\mathbf{M}$. Assuming $tanh$ is used as output activation function, $tanh^{-1}o_t$ is collected for each time step into a target collection matrix, $\mathbf{T}$.

If $\mathbf{W}^{out}$ is the weights from all the nodes in the network to the output nodes, we want to solve the equation $\mathbf{MW}^{out} = \mathbf{T}$. To solve for $\mathbf{W}^{out}$ we use the Moore-Penrose pseudoinverse; $\mathbf{W}^{out} = \mathbf{M}^{+}\mathbf{T}$.

Note that when the desired output is known, the network will learn the input-output function after only one presentation of the training sequence.

The input of the inverse model is the current state together with the desired next state, and the desired output is the desired motor command. The desired motor command is only known indirectly trough the desired position of the limbs. Generally, several motor commands may result in the same position of the limbs, and one does not want to bias the controller into choosing one specific solution. In sections 3.2 and 3.5 it is explained how an estimate of the desired motor command is used for teacher forcing and when generating the target collection matrix.

3 Learning the Inverse Model by Imitation

Our agent is implemented as a simple stick-man-simulator. After it has learned to imitate one movement, we want it to be able to imitate any movement presented by the demonstrator without further training. The input to the control system is always the current state and the desired next state (which is provided by the demonstrator). The goal is thus to learn the function mapping the current state and desired next state to the motor command, preferably with minimal effort.

3.1 The Movement Data

In the implementation of the experiments, we used a recording of the dance to the song YMCA by The Village People (se figure 1). The movement data was gathered with a Pro Reflex 3D motion tracking system by Axel Tidemann [13].

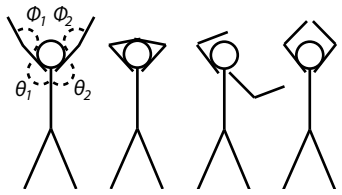


Fig. 1. The Movement is described with four angles; the two shoulder abduction angles $\theta 1$ and $\theta 2$ and the elbow extension angles $\phi 1$ and $\phi 2$.

The movement of each arm was described in two degrees of freedom, the angle between the arm and the body, i.e., the abduction angle θ, and the angle between the under- and upper arm, i.e., the extension in the elbow ϕ. Hence, the simulated robot was described by 4 degrees of freedom.

The YMCA movement was represented as a sequence of states, where each state t represents the four desired joint angles at time step t. The movement was manipulated in different ways to generate various training and testing sequences.

The goal of the control system is to produce motor commands that generate a movement where the state generated in each time step is as close as possible to the corresponding state in the desired sequence of states.

3.2 The Architecture

The control architecture that underlie the experiments is shown in figure 2. It consists mainly of an inverse model, but to achieve this model, a feedback controller is included during training.

At each time step t the control system receives as input the *desired next state* (i.e., the joint angles at time step $t+1$ in the training sequence) and the *current state*, and outputs a *motor command*, u.

During the training phase the feedback controller translates analytically the difference between the *desired current state* and the *actual current state* (i.e., state error) to a motor command, u_{error}. This motor error, the error done by the control system in the previous time step, is used to adjust the motor command for the current time step. This works as an approximation to teacher forcing because the only connection from the output nodes back to the network is through the plant, providing the current state input at the next time step. How much the feedback controller is able to influence the motor command depends on the *feedback gain*, K, by letting $u_{feedback} = K * u_{error}$. Note that during testing $K = 0$. The motor command $u_{feedback}$ is added to the motor command produced by the inverse model, and the result is sent to the plant, i.e., the robot simulator.

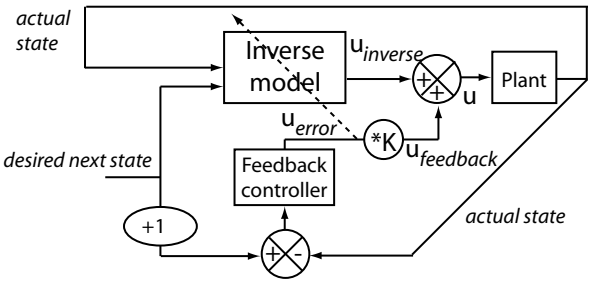


Fig. 2. The control architecture. Delays are shown with ellipses, i.e., the desired next state is delayed one time step, now representing the desired current state, before given as input to the feedback controller. An arrow across a model represents training, and u is a motor command.

3.3 Hypotheses

First, we need to verify that the control system presented is able to imitate novel movements when trained to perform one movement. This would imply that the system has learned at least parts of the function that computes the motor command needed to move the agent from it's current position to the desired next position. Second, we investigate further what properties the training movement must possess in order to make the system generalize to any movement in the state space. Our aim can be conveyed through the following tree hypotheses:

Hypothesis 1: *When training on imitating one movement, the control system does not only learn to mimic that movement, but learns at least parts of the function mapping a current and a desired state to a motor command, which will make it able to imitate other movements without training.*

Hypothesis 2: *In order to learn to control one particular degree of freedom, it has to be included in the training movement.*

Hypothesis 3: *When trained on synchronous movements, i.e., the movement of the two arms are equivalent, mirroring each other, the control system is only able to imitate synchronous movements. Training on movements where the limbs follow different trajectories is necessary in order to make the control system generalize to all movements.*

3.4 Experiments

To generate different training and testing sequences the YMCA movement was manipulated in different ways. The movements were **YMCA** (the whole YMCA movement), **Y** (only the Y movement, moving back to start position by reversing the Y motion), **Y pure** (a manipulated version of the Y movement, where all movement in elbow angle is removed), **YM** (only the YM movement, moving back by reversing the YM sequence), **right arm mirror** (both arms does the movement of the right arm in the YMCA movement, making the arms mirror

each other) and **left arm mirror** (similar to *right arm*, but now both arms moves as the left arm in the YMCA movement).

3.5 Training

The training procedure was organized in epochs and cycles, where one cycle is one full temporal presentation of the training motion. In each epoch we ran seven cycles. First, we re-initialized the network by setting the internal states of the network to zero and run one cycle without updating the output weights. Subsequently, the training sequence was presented five times with enabled learning. The output connections were then adapted after each complete cycle. A final cycle was used to estimate the performance error on the training sequence while learning was disabled. The training was run for 150 epochs.

Multiple training epochs was needed because perfect teacher forcing could not be provided. To make the estimate of the desired motor command, u, as good as possible, the feedback controller should provide less influence as the inverse model gets more accurate. This was ensured by decreasing the feedback gain, K, by 10% each epoch.

The output from the feedback controller was also used when calculating the target collection matrix. Because we had batch learning, the motor command was stored and the target motor command u_{target} was calculated by using the u_{error} provided in the next time step, $u^t_{target} = u^t + u^{t+1}_{error}$ (because u^t_{error} reflects the error done at time step $t-1$).

The inverse model had 8 input nodes, 1000 nodes in the internal layer and 4 output nodes. The ESN had spectral radius $\alpha = 0.1$ (determining the length of the memory with increasing $\alpha \in [0, 1]$) and noise level $v = 0.2$ (effectively adding 10% noise to the internal state of the network). The inputs and outputs where scaled to be in the range [-1,1]. Normal-distributed noise with standard deviation 0.01 was added to the sensory signal from the plant. The system was implemented in MatLab, including the simple stick-man-simulator used as plant.

3.6 Testing

After the inverse model was trained to imitate one movement, we wanted to test whether it could imitate other movements without training. This was done by changing the desired sequence and run the network for one additional epoch with only two cycles, the initialization cycle and the evaluation cycle.

During training the feedback gain was decreased to ~ 0, and the feedback controller was thus removed from the control system during testing.

4 Results

This section summarizes the results of the experiments. The results verifying hypotheses 1, 2 and 3 are described in section 4.1, 4.2 and 4.3 respectively. We have illustrated the results by plotting the shoulder abduction angle, θ against

the elbow extension angle, ϕ for each arm. The temporal dimension is not plotted because all discrepancies between actual and desired movement could be seen as spatial errors, the timing turned out to not be a problem in any of the experiments. Figure 3 shows a plot of the whole YMCA movement where the different parts of the movement is separated by the use of different markers.

4.1 Does the Control System Generalize?

The initial experiment was to train the control system to imitate the whole YMCA movement and see if it was able to produce the correct movements when tested on the other movements described in section 3.4. We also tested whether the control system would generalize to different speeds. The control system managed all these tests without significantly more error than when imitating the trained movement. We conclude that when trained with the whole YMCA movement, the control systems learned the complete mapping from current- and desired state to motor command in the state space.

4.2 Training All Degrees of Freedom

Does all degrees of freedom, used in the testing sequence, need to be included in the training sequence? To test this hypothesis the control system was trained on *Y pure* and tested on *YM*, see figure 4. The control system is clearly not able to utilize the joint it has not trained to use. To find out how small perturbations in the joint is sufficient for generalization, we tried training on *Y* and testing on *YMCA*. As figure 5 illustrates, small angular changes in the joint during training, makes it possible to generalize to larger changes during testing, but not large enough to perform *YMCA* without large errors.

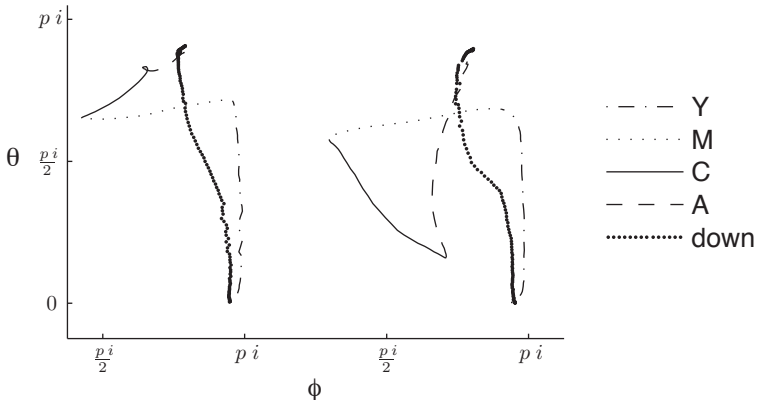


Fig. 3. The whole YMCA movement. The shoulder abduction angle, θ is plotted against the elbow extension angle, ϕ for each arm. The left graph shows the movement of the right arm and the right graph the movement of the left arm. Each part of the movement is plotted with a different marker to distinguish them from each other.

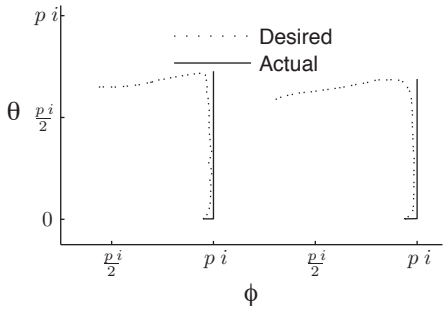


Fig. 4. The figure shows the trajectory produced by the two arms when the control system is trained on *Y pure*, and tested on *YM*. In the training sequence, *Y pure*, there is no movement in the elbow joints, and thus the network is not able to utilize these during testing.

4.3 Synchronous and Asynchronous Movements

To test whether the system can generalize to asynchronous movements when trained with a pure synchronous movement, we used *right arm mirror* and *left arm mirror* as training sequences and tested on *YMCA* (see figure 6).

The system clearly does not generalize to asynchronous movements; the movement of the arms were more or less symmetric even though the test sequence is not. The movement of each arm was an average of the movements of the two arms in the desired sequence. As a consequence the results are practically identical when the control system was trained with *right arm mirror* compared to when trained with *left arm mirror*.

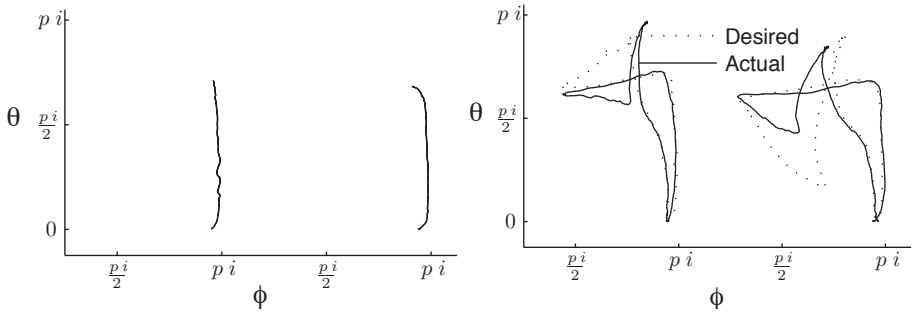


Fig. 5. The left figure illustrates the *Y* movement. Notice that there is hardly any motion in the elbow. The figure to the right shows the result when the control system trained on movement *Y* is tested on movement *YMCA*. The control system is able to generate more motion in the elbow joints than it learned during training. However, it is not able to produce the *YMCA* movement without large errors.

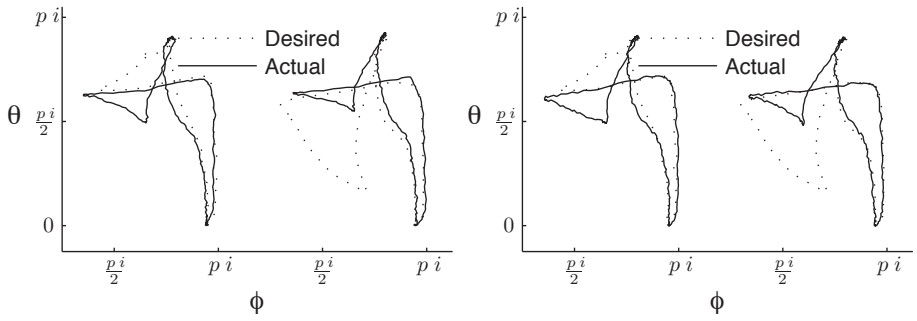


Fig. 6. The figures illustrates the results when trying to do an asynchronous movement when trained to do a synchronous one. The control system produces a synchronous movement that is the average of the desired movement of the two arms. In the figure to the left the system was trained on *left arm mirror* and in the right figure, *right arm mirror*. Both were tested on *YMCA*.

Remember that the opposite problem of imitating a synchronous movement when trained with an asynchronous movement does not pose any problem. When trained with the asynchronous movement *YMCA*, the system was able to generate all the movements without difficulty.

5 Discussion and Conclusion

Our aim was to use imitation to efficiently learn the inverse kinematics model of our robot simulator. We showed that when trained to imitate one movement, the control system has not only learned to imitate that particular movement, but is able to imitate novel movements without further training. This means that the system has learned at least parts of the desired inverse model, verifying hypothesis 1.

Our second hypothesis envisages that in order to learn to control one particular degree of freedom, it has to be included in the training movement. We showed this to be true. In addition, our results suggest that the control system does not have to train on the whole range of motion for each degree of freedom in order to generalize to all movements. This is important when we want to train the inverse model with minimal amount of effort.

Hypothesis 3 suggests that asynchronous movements are harder than synchronous movements, and that the control system will not be able to produce different motor commands for the two arms if it has not been trained to do so. For humans it is indeed true that it is easier to move the limbs synchronously. It is still very interesting that we get the same results for this control system, and interesting to see that a system trained to produce a synchronous movement and asked to generate an asynchronous movement provides the best solution it is

able to, namely the average between the desired movement of the left and right arm.

Our findings suggests that imitation may be used as an efficient method to learn the inverse model, because one can choose the training sequence optimally, as opposed to exploration without guidance. This conclusion is supported by Rolf et. al. who suggests the use of goal directed exploration in contrast to motor babbling [14].

Further work should include more complex movements with larger degrees of freedoms where one target position can be reached through different motor commands. In addition more systematic evaluation of efficient training movements should be conducted.

References

1. Belardinelli, A., Pirri, F.: Bottom-up gaze shifts and fixations learning by imitation. IEEE Trans Syst Man Cybern B Cybern **37**(2) (2007) 256–271
2. Wood, M.A., Bryson, J.J.: Skill acquisition through program-level imitation in a real-time domain. IEEE Trans Syst Man Cybern B Cybern **37**(2) (2007) 1083–4419
3. Meltzoff, A.N., Moore, M.K.: Explaining facial imitation: A theoretical model. Early Development and Parenting **6** (1997) 179–192
4. Demiris, Y., Dearden, A.: From motor babbling to hierarchical learning by imitation: a robot developmental pathway. In: Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems. (2005) 31–37
5. von Hofsen, C.: An action perspective on motor development. TRENDS in Cognitive Sciences **8**(6) (2004) 266–272
6. Wolpert, D.M., Miall, R.C., Kawato, M.: Internal models in the cerebellum. Trends in Cognitive Sciences **2**(9) (1998)
7. Shadmehr, R., Krakauer, J.W.: A computational neuroanatomy for motor control. Experimental Brain Research **185**(3) (2008) 359–381
8. Davidson, P.R., Wolpert, D.M.: Widespread access to predictive models in the motor system a short review. Journal of Neural Engineering **2**(3) (2005) 313–319
9. Kawato, M., Gomi, H.: The cerebellum and vor/okr learning models. TINS **15**(11) (1992)
10. Mehta, B., Schaal, S.: Forward models in visuomotor control. Journal of Neurophysiology **88**(2) (2002) 942–953
11. van der Smagt, P., Hirzinger, G.: The cerebellum as computed torque model. In Howlett, R., Jain, L., eds.: Fourth International Conference on Knowledge-Based Ingelligent Engineering Systems & Applied Technologies. (2000)
12. Jaeger, H.: A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach. Techreport, Fraunhofer Institute for AIS (April 2008)
13. Tidemann, A., Öztürk, P.: Self-organizing multiple models for imitation: Teaching a robot to dance the YMCA. In: IEA/AIE. Volume 4570 of LNCS., Springer (June 2007) 291–302
14. Rolf, M., Steil, J.J., Gienger, M.: Efficient exploration and learning of whole body kinematics. In: IEEE 8th International Conference on Development and Learning. (2009)

LEARNING TO IMITATE YMCA WITH AN ECHO STATE NETWORK

D

Author:
Rikke Amilde Løvlid

Abstract: When an echo state network with feedback connections is trained with teacher forcing and later run in free mode, one often gets problems with stability. In this paper an echo state network is trained to execute an arm movement. A sequence with the desired coordinates of the limbs in each time step is provided to the network together with the current limb coordinates. The network must find the appropriate angle velocities that will keep the arms on this trajectory. The current limb coordinates are indirect feedback from the motor output via the simulator. We do get a problem with stability in this setup. One simple remedy is adding noise to the internal states of the network. We verify that this helps, but we also suggest a new training strategy that leeds to even better performance on this task.

Published in:
Artificial Neural Networks and Machine Learning - ICANN 2012, *Lecture Notes in Computer Science*, pages 507-514.

Learning to Imitate YMCA with an Echo State Network

Rikke Amilde Løvlid

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

Abstract. When an echo state network with feedback connections is trained with teacher forcing and later run in free mode, one often gets problems with stability. In this paper an echo state network is trained to execute an arm movement. A sequence with the desired coordinates of the limbs in each time step is provided to the network together with the current limb coordinates. The network must find the appropriate angle velocities that will keep the arms on this trajectory. The current limb coordinates are indirect feedback from the motor output via the simulator. We do get a problem with stability in this setup. One simple remedy is adding noise to the internal states of the network. We verify that this helps, but we also suggest a new training strategy that leeds to even better performance on this task.

1 Introduction

Echo state networks (ESNs) are a fairly new type of recurrent neural networks that have shown great potential for solving non-linear, temporal problems. The basic idea is to transform the low dimensional temporal input into a higher dimensional *echo state*, and then train the output connection weights to make the system output the desired information. Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks [1].

In this paper we use an ESN to compute the inverse kinematics of an arm movement. The network is trained to output the joint angle velocities that will move the arm from the current position to the next desired position. After moving the arm, the resulting position is used as input to the network as the current position in the next time step.

A know problem with ESNs are unstable output when the network is trained to predict one step ahead with teacher forcing and is later run in a generative mode, looping its output back into the input. Models having feedback connections in general might have this problem, even when they are driven by external input, as is the case in our setup. The reason for these potential instabilities is that even if the model can make a pretty accurate one step prediction, going through the feedback loop small errors get amplified and can make the output diverge from the desired output.

As will be shown in section 7, we do get a problem with stability. A classical remedy is adding noise to the internal states during training [1]. This makes the network learn the desired next target from the neighborhood of the current state. Adding noise to the network helps in our case too, but is not enough to solve the problem. Other suggestions include using ridge regression [2], pruning the output weights [3] or updating the weights based on the particle swarm optimization algorithm [4]. We suggest a new strategy, namely gradually adapting the target used when computing the output weights.

2 Learning to Imitate YMCA

In the implementation of the experiments, we used a recording of the dance to the song YMCA by The Village People. The movement data was gathered with a Pro Reflex 3D motion tracking system by Axel Tidemann [5]. The movement of each arm was described in six degrees of freedom (DOFs), the x, y and z coordinates of the elbow position relative to the shoulder and the wrist position relative to the elbow. The coordinates was normalized to be in the interval $\langle -1, 1 \rangle$. The movement sequence was divided into 312 steps and one repetition of this movement was used as the basic training sequence.

The robot simulator had 3 DOFs in each shoulder and 1 DOF in each elbow joint, totally 8 DOF. The arms are moved by specifying the angular velocity for each DOF in each time step.

The task is to make this simulator execute the YMCA trajectory, which means calculating the angular velocities (i.e. motor commands) for each DOF in each time step based on the actual current position and the next position in the desired trajectory, both represented as cartesian coordinates as described above. This is a non-linear, one-to-many mapping.

In this paper the ESN will only try to execute the trajectory which it has trained on, but ultimately we will want it to generalize to other movements, which means learning the inverse model of the motor apparatus. We have earlier done experiments in 2D where we investigated the benefit of learning the inverse model by training on one movement with certain properties [6]. When trying to repeat those experiments in 3D, we had problems learning the training movement itself. This paper proposes as new training method for ESNs, which makes us able to learn the training movement i 3D.

3 The Original Training Algorithm

A general echo state network is illustrated in figure 1.

The activation of the internal nodes is updated according to

$$\mathbf{x}(t) = f(\mathbf{W}^{in}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1) + \mathbf{W}^{back}\mathbf{y}(t-1)) + v(t-1), \tag{1}$$

where f is the node's activation function, and v are white Gaussian noise. The output of the network is computed according to

$$\mathbf{y}(t) = f^{out}(\mathbf{W}^{out}(\mathbf{u}(t), \mathbf{x}(t))). \tag{2}$$

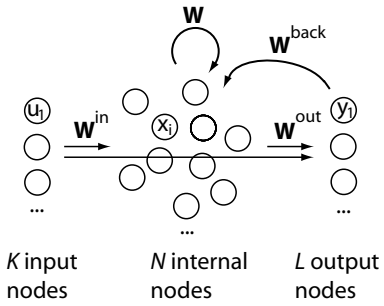


Fig. 1. The figure illustrates a basic ESN.

A general task is described by a set of input and desired output pairs, [$\langle u(1),\ y_{target}(1)\rangle$, $\langle u(2),\ y_{target}(2)\rangle$, ..., $\langle u(T),\ y_{target}(T)\rangle$]. The solution is a trained ESN whose output $\mathbf{y}(t)$ approximates the teacher output $\mathbf{y}_{target}(t)$ when the ESN is driven by the training input $\mathbf{u}(t)$. Generating this solution ESN is done in three steps.

First, a random RNN with the Echo State property is generated [1]. Second, the training sequence is run through the network once. If there are feedback connections, teacher forcing is used, meaning $y(t)$ is replaced by $y_{target}(t)$ when computing $x(t+1)$ and $y(t+1)$. After the first T_0 time steps, which are used to wash out the initial transient dynamics, the state of each input node and internal node in each time step is stored in a state collection matrix, $\mathbf{M}$. Assuming $tanh$ is used as output activation function, $tanh^{-1}y_{target}(t)$ is collected row-wise into a target collection matrix $\mathbf{T}$. Equation 2 can then be written:

$$\mathbf{T} = \mathbf{M}(\mathbf{W}^{out})^T. \tag{3}$$

Third, the output weights are computed by using the Moore-Penrose pseudo-inverse to solve equation 3 with regard to $\mathbf{W}^{out}$:

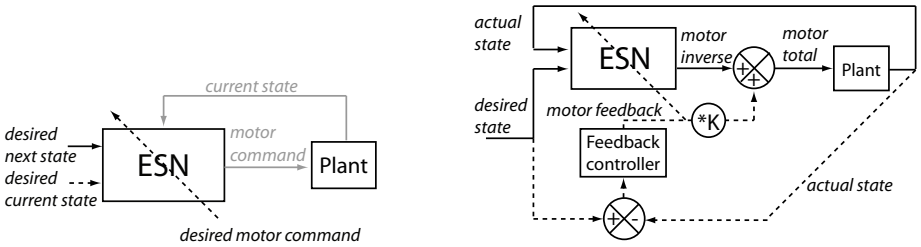
$$(\mathbf{W}^{out})^T = \mathbf{M}^{+}\mathbf{T}. \tag{4}$$

4 The new Proposed Training Strategy

In the original training algorithm the training sequence is run through the network once. The output weights are updated based on the target collection matrix and the state collection matrix as shown in equation 4. We suggest running the training sequence through the network several times. In each cycle the weights are calculated based on the state collection matrix and something in between the target and the actual output from the inverse model. The target used when computing $\mathbf{W}^{out}$ in cycle i is

$$\mathbf{y}^{i}_{target}(t) = \beta\mathbf{y}_{target}(t) + (1-\beta)\mathbf{y}(t). \tag{5}$$

We hypothesis that this new proposed training method will improve performance during testing.



(a) True Target (b) Feedback Controller

Fig. 2. Two architectures were used in the experiments; (a) the inverse model is trained with true target information and teacher forcing, and (b) a feedback controller is used both for estimating the motor error and providing teacher forcing. In the latter, the feedback gain, K, decides how much the feedback controller influences the final motor command. K is gradually decreased from 1 to 0 during several rounds of training. The dotted lines are only used during training whereas the grey lines are only used during testing.

5 The Architectures

The experiments were conducted with two different architectures. In the first architecture we calculate the desired motor commands with use of the inverse function given in the paper by Tolani and Badler [7]. True teacher forcing was provided during training by replacing the current position from the simulator with the next desired position from the training sequence before using it as input to the network. In this architecture the plant is only used during testing. The architecture is illustrated in figure 2(a) and will be referred to as the *true target* architecture.

In the second architecture we recognize that the desired joint angles are generally not available. What is know is the desired arm positions. Also, there are several joint angle configurations that represent the same arm position. The feedback-error-learning scheme proposed by Kawato [8] is designed to handle these issues. In this architecture a feedback controller transforms the error in position to an error in motor commands. This estimated motor error is used both as an estimate of the target and for teacher forcing. The architecture is drawn in figure 2(b) and will be referred to as the *feedback controller* architecture.

In order to make the inverse model learn and the feedback controller redundant, the feedback gain, K, was linearly decreased from 1 to 0 during 10 epochs. During testing the feedback gain was 0.

6 Experiments

The main objective is to investigate whether the new training method outperforms the original training method. In addition we study the effect of adding noise to the internal network, and try to improve the performance of the original

method by adding repetitions of the YMCA movement in the training sequence. This can be summarized in four experiments:

Experiment 1: *Original training method without noise during training.*
Experiment 2: *Original training method with noise in the network.*
Experiment 3: *Original training method with noise and several repetitions of the YMCA movement in the training sequence. The experiments when using 5 and 10 repetitions are referred to as 3a and 3b respectively.*
Experiment 4: *New proposed training method with $\beta = 0.1$. There is still noise in the network, and the training sequence consists of only one repetition of the YMCA movement.*

When implementing the ESN we used the simple matlab toolbox provided by Jaeger [9]. The spectral radius was 0.5 and tanh was used as output function. When noise was included in the reservoir, the noise level was set to 0.2, effectively adding 10% noise to the internal states. All other network parameters used were the default in the toolbox. Gaussian noise with mean 0 and standard deviation 0.01 was added to the output from plant.

In experiment 4 training time increases as β decreases because additional passes of the training sequence must be done to make the target used converge towards the actual target. We therefor want β to be as high as possible. To make the network learn the training sequence in the architecture with the feedback controller, we had to use $\beta = 0.1$ (or lower). For this value of β, 50 rounds of training is sufficient for convergence, which meant 5 training cycles for each value of the feedback gain when using the feedback controller.

To evaluate the results we use the Root Mean Square Error (RMSE) normalized over the range of the output values, $NRMSE(\mathbf{y}, \mathbf{y}_{target}) = \frac{\sqrt{MSE}}{y_{max} - y_{min}}$. The NRMSE for each run was averaged over all time steps and DOFs.

7 Results

Each experiment was repeated 50 times. Figure 3 compares the results in a box and whisker plot and table 1 contains the average NRMSE with standard deviation for each experiment. Note that when training with a feedback controller, only the position error during testing is interesting. The motor error cannot be computed because we do not have a correct motor command to compare with. Also the training error is not interesting, since in this setup there is a gradual transition from training to testing as the feedback gain is reduced. In the true target architecture however, training and testing means teacher forcing on or off, i.e. one step prediction or generative mode respectively.

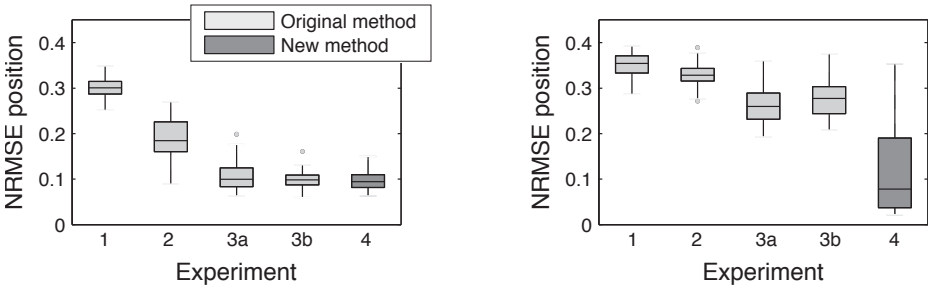
To illustrate the effect on the actual movement and visualize the problem with instability, figure 4 compares the actual- with the desired positions of the right arm from one run of each of the experiments 1 (original), 2 (original w/noise) and 4 (new method w/noise) in the true target architecture.

Table 1. The average NRMSE and variance for each experiment after 50 runs.

	True Target				*Feedback Controller*
	Position train	Motor train	Position test	Motor test	Position test
Experiment 1	0.0033	0.0007	0.3000	0.3228	0.3490
Original Method	4.94E-10	1.47E-09	4.16E-04	1.99E-04	7.0786e-04
Experiment 2	0.0092	0.0053	0.1863	0.0991	0.3289
Orig. Met. w/noise	1.10E-07	2.58E-08	2.16E-03	5.50E-04	6.6268e-04
Experiment 3a	0.0070	0.0163	0.1056	0.1539	0.2587
Orig. Met. 5 rep.	2.28E-09	2.14E-08	8.92E-04	8.62E-04	0.0013
Experiment 3b	0.0068	0.0156	0.0994	0.1519	0.2770
Orig. Met. 10 rep.	8.10E-10	1.60E-08	3.33E-04	3.30E-04	0.0013
Experiment 4	0.0070	0.0039	0.0973	0.0476	0.1164
New Met. w/noise	1.21E-07	5.47E-09	4.96E-04	1.37E-04	0.0084

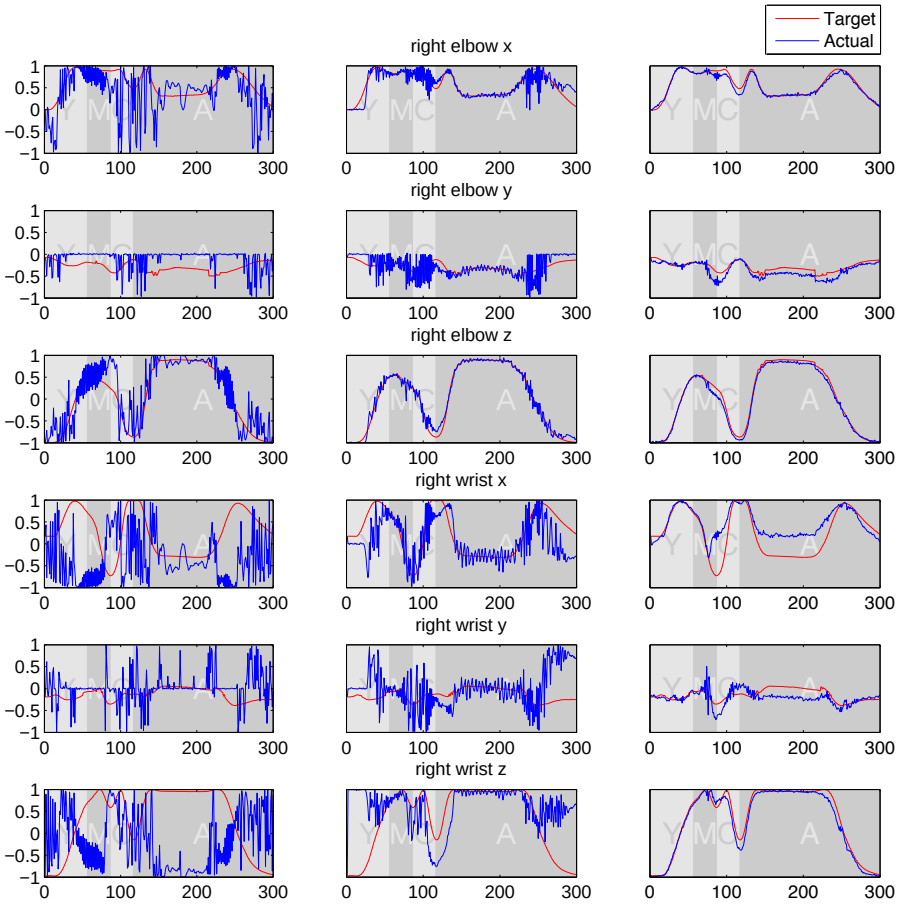
In the simplest case, with true target, adding noise to the internal networks improves the testing results. This is seen by comparing the results of experiment 1 and 2 in figure 3 and table 1. However, when using the feedback controller, the improvement is hardly visible.

Training the network with the YMCA movement repeated several times makes significant improvement, especially in the true target architecture. In the true target architecture we are able to get just as good results with the original method adding more repetitions as using the new training method on one repetition of the YMCA movement. However, when training with a feedback controller, the new method clearly outperforms the original.



(a) True Target (b) Feedback Controller

Fig. 3. Box and whisker plot for 50 runs of each of the experiments. On each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually.



(a) Original Method (b) Orig. Method w/noise (c) New Method w/noise

Fig. 4. The desired versus actual right arm positions during testing for one average run of each of the experiments 1, 2 and 4 with the true target architecture. The average position error (both arms) for these selected runs was 0.300, 0.193 and 0.091 for experiment 1,2 and 3 respectively.

8 Discussion and Conclusion

In the original training method, the training sequence is run through once (for each value of the feedback gain). The states of the network are collected in each time step, and the output weights are calculated from this state collection and the target sequence. The original method finds the best fitted linear transformation from the reservoir to the target states. The new method proposed in

this paper runs the training sequence through several times, each time calculating the output weights by finding the best fitted linear transformation from the reservoir to a sequence between the actual output states of the ESN and the target output states.

When the target sequence is known, the novel training strategy is no better than using the original method on an extended version of the training sequence. Adding several repetitions of the temporal pattern is a common strategy when there is noise in the training data, which is almost always the case. However, when training the ESN with the feedback controller, the novel training strategy still outperforms the original. We hypothesize that this is because the quality of the target estimate and the teacher forcing becomes better as the network performs with less error. This makes it beneficial to carry out the learning gradually. It would be interesting to test the proposed method on a problem where the error calculation can be done more accurately as the error decreases.

Further work could also test this new method on benchmark problems like generation of the figure eight [2] or a chaotic attractor like the Mackey-Glass system [10]. The focus would then be whether the novel method proposed here, would be faster than using the original training method on a longer sequence. Preliminary studies indicate that this might in fact be true.

References

1. Jaeger, H.: A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems (2002)
2. Wyffels, F., Schrauwen, B., Stroobandt, D.: Stable output feedback in reservoir computing using ridge regression. In: Proceedings of the 18th international conference on Artificial Neural Networks, Part I. ICANN '08, Berlin, Heidelberg, Springer-Verlag (2008) 808–817
3. Dutoit, X., Schrauwen, B., Campenhout, J.V., Stroobandt, D., Brussel, H.V., Nuttin, M.: Pruning and regularization in reservoir computing: a first insigth. In: Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN 2008). (2008) 1–6
4. Song, Q., Feng, Z., Lei, M.: Stable training method for echo state networks with output feedbacks. In: Networking, Sensing and Control (ICNSC), 2010 International Conference on. (april 2010) 159 –164
5. Tidemann, A., Öztürk, P.: Self-organizing multiple models for imitation: Teaching a robot to dance the YMCA. In: IEA/AIE. Volume 4570 of LNCS., Springer (June 2007) 291–302
6. Løvlid, R.A.: Learning motor control by dancing ymca. In: IFIP Advances in Information and Communication Technology. (2010) 79–88
7. Tolani, D., Badler, N.I.: Real-time inverse kinematics of the human arm. Presence **5**(4) (1996) 393–401
8. Kawato, M.: Feedback-error-learning neural network for supervised motor learning. In Eckmiller, R., ed.: Advanced Neural Computers. Elsevier (1990) 365–372
9. Jaeger, H.: Simple toolbox for esns url: reservoir-computing.org/software (2009)
10. Jaeger, H.: The echo state approach to analysing and training recurrent neural networks. Technical report, GMD (2001)

A Novel Method for Training an Echo State Network with Feedback Error Learning

E

Author:
Rikke Amilde Løvlid

Abstract:
Echo state networks are a relatively new type of recurrent neural net- works which have shown great potentials for solving nonlinear, temporal problems. The basic idea is to transform the low dimensional temporal in- put into a higher dimensional state, and then train the output connection weights to make the system output the desired information. Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks.

This paper investigates using an echo state network to learn the inverse kinematics model of a robot simulator with feedback-error-learning. In this scheme teacher forcing is not perfect, and joint constraints on the simulator makes the feedback error inaccurate. A novel training method which is less influenced by the noise in the training data is proposed and compared to the traditional ESN training method.

Published in:
Advances in Artificial Intelligence

Hindawi Publishing Corporation
Advances in Artificial Intelligence
Volume 2013, Article ID 891501, 9 pages
http://dx.doi.org/10.1155/2013/891501

Research Article

A Novel Method for Training an Echo State Network with Feedback-Error Learning

Rikke Amilde Løvlid

Department of Computer and Information Science, Norwegian University of Science and Technology, Sem Sælands vei 7-9, 7491 Trondheim, Norway

Correspondence should be addressed to Rikke Amilde Løvlid; rikke-amilde.lovlid@ffi.no

Received 31 May 2012; Revised 10 December 2012; Accepted 19 February 2013

Academic Editor: Ralf Moeller

Echo state networks are a relatively new type of recurrent neural networks that have shown great potentials for solving non-linear, temporal problems. The basic idea is to transform the low dimensional temporal input into a higher dimensional state, and then train the output connection weights to make the system output the target information. Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks. This paper investigates using an echo state network to learn the inverse kinematics model of a robot simulator with feedback-error-learning. In this scheme teacher forcing is not perfect, and joint constraints on the simulator makes the feedback error inaccurate. A novel training method which is less influenced by the noise in the training data is proposed and compared to the traditional ESN training method.

1. Introduction

A recurrent neural network (RNN) is a neural network with feedback connections. Mathematically RNNs implement dynamical systems, and in theory they can approximate arbitrary dynamical systems with arbitrary precision [1]. This makes them "in principle promising" as solutions for difficult temporal tasks, but in practice, supervised training of RNNs is difficult and computationally expensive.

Echo state networks (ESNs) were proposed as a cheap and fast architectural and supervised learning scheme and are therefore suggested to be useful in solving real problems [2]. The basic idea is to transform the low dimensional temporal input into a higher dimensional *echo state*, and then train the output connection weights to make the system output the desired information. The idea was independently developed by Maass [3] and Jaeger [4] as liquid state machine (LSM) and echo state machine (ESM), respectively.

LSMs and ESMs, together with the more recently explored Backpropagation Decorrelation learning rule for RNNs [5], are given the generic term reservoir computing [6]. Typically large, complex RNNs are used as reservoirs, and their function resembles a tank of liquid. One can think of the input as stones thrown into the liquid, creating unique ripples that propagate, interact, and eventually fade away. After learning how to read the water's surface, one can extract a lot of information about recent events, without having to do the complex input integration. Real water has successfully been used as a reservoir [7].

Because only the output weights are altered, training is typically quick and computationally efficient compared to training of other recurrent neural networks.

We are investigating how to use an ESN to learn internal models of a robot's motor apparatus. An internal model is a system that mimics the behavior of a natural process. In this paper we will talk about inverse models, which transform preplanned trajectories of desired perceptual consequences into appropriate motor commands.

The inverse model is often divided into a kinematic and a dynamic model. An inverse kinematic model transforms a trajectory in task space (e.g., cartesian coordinates) to a trajectory in actuator space (e.g., joint angles), and an inverse dynamic model transforms the joint space trajectory into the sequence of forces that will actually move the limbs. The

robot simulator in our experiments is controlled by the joint angle velocities directly, thus we are only concerned with kinematics.

It is common to use analytical internal models, and deriving such a model for our simulator would be easy. Despite this, we want to explore using an ESN as an inverse model, because as robots become more complex, with springy joints, light limbs and many degrees of freedom, acquiring analytical models will become more and more difficult [8]. Oubbati et al. also argue that substituting the analytical models with a recurrent neural networks might be beneficial in general, as it can make the inverse model more robust against noise and sensor errors [9].

To acquire an accurate inverse model through learning is, however, problematic, because the target motor commands are generally unavailable. What is known is the target trajectory in task space. Three schemas have been suggested for training the inverse model: directly by observing the effect of different motor commands on the controlled object [10], with a forward model as a distal teacher [11], or with an approach called feedback-error learning (FEL) [10]. Direct modeling was excluded because it cannot handle redundancies in the motor apparatus and therefore will not scale to real problems [11]. FEL was chosen over distal teacher because it is a natural extension of using an analytical model, and because it is biologically motivated due to its inspiration from cerebellar motor control [12]. Another advantage, which we will not exploit here, is that FEL can be used for control during learning.

The objective in this paper is to investigate how an ESN can be trained within this FEL scheme. The traditional ESN learning method falls short in this setup due to inaccurate teacher forcing and target estimation. We propose a novel training method, which is inspired by gradient decent methods and shows promising results on this problem. Preliminary studies of this training method can be found in a related work [13]. The current paper includes further studies of why this new method works so well.

2. Learning to Imitate YMCA

In this paper an ESN is trained to execute an arm movement on a simple robot simulator by computing the inverse kinematics of that movement. The ESN is only tested on the movement it was trained on, which means that we do not verify whether the ESN has actually learned the inverse model or merely to execute this particular trajectory. We have earlier investigated the benefit of learning the inverse model by training on one movement with certain properties [14]. Here we have a slightly more complex inverse problem and encountered a problem when trying to learn the training sequence itself. The solution to that problem is the main point in this paper.

2.1. Training Data. The movement data is a recording of the dance to the song YMCA by the Village People. It was gathered with a Pro Reflex 3D motion tracking system by Tidemann and Öztürk [15]. The system is able to track the position of fluorescent balls within a certain volume by using five infrared cameras. The sampling frequency of the Pro Reflex is 200 Hz. In the experiments we used every fourth sample, meaning the position trajectory consisted of 50 samples/sec, resulting in a sequence with 313 steps.

The tracking of the balls yields cartesian coordinates of the balls in three dimensions. The result was projected down to two dimensions, and the position of each arm was expressed as the x and z coordinates of the elbow relative to the shoulder and the wrist relative to the elbow. The coordinates were normalized to be in the interval $\langle -1, 1\rangle$. The position in each time step was thus represented by 8 signals, that is, $(x_{\text{elbow}}, z_{\text{elbow}}, x_{\text{wirst}}, z_{\text{wrist}})$ for each arm.

2.2. Simulator. For the simulations we used a fairly simple 2D simulator with four degrees of freedom (DOFs), one in each shoulder and one in each elbow. The simulated robot was controlled by the joint angle velocities directly, which means that the problem of translating the velocities into torques was not considered. The ESN was trained to output the joint angle velocities that would keep the elbows and wrists on the desired trajectory. The velocities were scaled to be in the interval $\langle -1, 1\rangle$ and will be referred to as the *motor commands*.

The range of motion was constrained to be between 0° and 180° for all 4 DOFs, and if the motor command implied moving the limb further, the limb stopped at the limit and the overshooting motor command was ignored.

The maximum joint angle velocity for each DOF was set to twice the maximum velocity registered in the recorded movement, which meant that a joint angle velocity equal to 1 moved the joint less than 180 degrees. Limited joint velocity is realistic, and it also makes large errors in motor commands lead to smaller position errors, making the movements look smoother.

2.3. Control Architecture. The ESN is trained to compute motor commands that will move the simulated arms from the current position to the next position in the target trajectory. The target motor commands needed for training are not available; what is available is the target positions.

The FEL scheme, illustrated in Figure 1, includes a feedback controller that estimates the error in motor command from the position error. The motor error computed by the feedback controller is used both to train the ESN and to adjust the motor command from the inverse model before it is sent to the arm simulator. In the current setup the transformation from position error to motor error is simple enough to be done analytically, but using the result will still not be perfect as the simulator is noisy and the calculation does not take into consideration any excess motor commands that were potentially ignored if the limbs were moved to their limits.

How much influence the feedback controller has on the final motor command is regulated by the feedback gain, K. To facilitate learning and force the feedback controller to become redundant, the feedback gain was linearly reduced from 1 to 0 during several rounds of training.

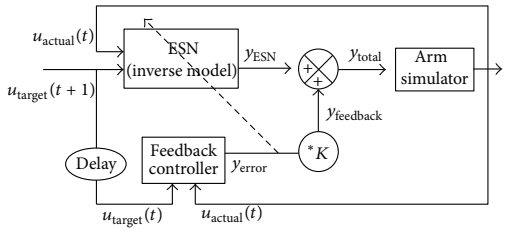


Figure 1: The figure illustrates the feedback-error-learning (FEL) architecture used to training the ESN. The input to the ESN is the actual position at the current time step ($u_{actual}(t)$) and the next position in the target position trajectory ($u_{target}(t+1)$). The ESN learns to calculate the motor command which will move the simulated arms from the current position to the next position in the target trajectory. The motor command from the ESN is called y_{ESN} and is adjusted by the motor command from the feedback controller, $y_{feedback}$, before it is used to move the simulated arms. The feedback controller estimates the error of this total motor command (y_{error}) by comparing the resulting position with the corresponding target position. This error is used to train the ESN and to compute the feedback motor command in the next time step. The feedback gain, K, determines how much the feedback controller can influence the total motor command.

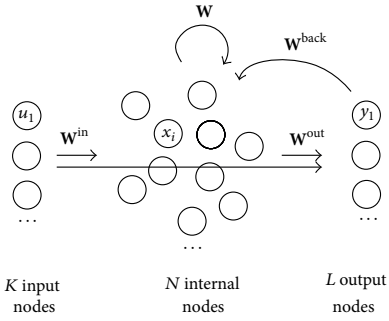


Figure 2: The figure illustrates a basic ESN.

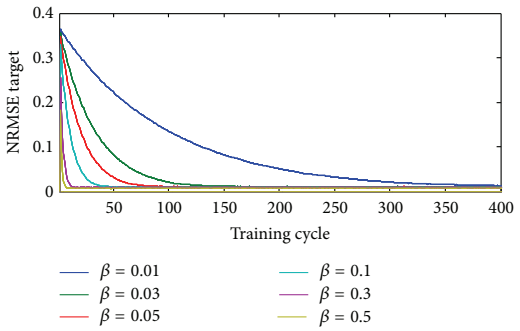


Figure 3: The plot shows the difference between the true target and the used target in each training cycle for different values of β when target estimation and teacher forcing are perfect. The result is used to deduce how many extra cycles of training are needed for different values of β. Note that with $\beta = 1$, the used target and the true target will be equal from the start, and only one cycle of training is needed.

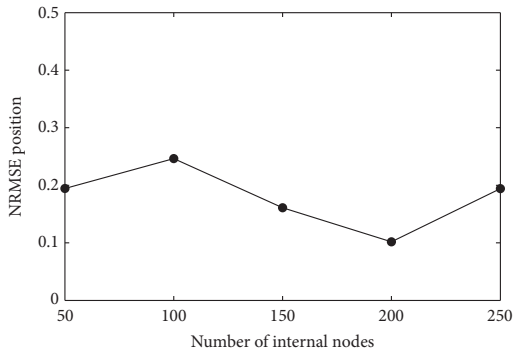


Figure 4: To determine the optimal reservoir size, ESNs with different numbers of internal nodes were trained with the original training method within the FEL scheme. The YMCA movement was repeated 5 times in the training sequence, and the internal noise level was 0.02. The figure shows the mean position errors during testing for 10 repetitions of each experiment.

3. Training an Echo State Network

A basic echo state network is illustrated in Figure 2. The activation of the internal nodes is updated according to

$$\mathbf{x}(t) = f\left(\mathbf{W}^{\text{in}}\mathbf{u}(t) + \mathbf{W}\mathbf{x}(t-1) + \mathbf{W}^{\text{back}}\mathbf{y}(t-1)\right) + v(t-1), \quad (1)$$

where f is the node's activation function, and v are white Gaussian noise. The output of the network is computed according to

$$\mathbf{y}(t) = f^{\text{out}}\left(\mathbf{W}^{\text{out}}(\mathbf{u}(t), \mathbf{x}(t))\right). \quad (2)$$

A general task is described by a set of input and desired output pairs, $[\langle \mathbf{u}(1), \mathbf{y}_{\text{target}}(1)\rangle, \langle \mathbf{u}(2), \mathbf{y}_{\text{target}}(2)\rangle, \ldots, \langle \mathbf{u}(T), \mathbf{y}_{\text{target}}(T)\rangle]$, and the solution is a trained ESN whose output $\mathbf{y}(t)$ approximates the teacher output $\mathbf{y}_{\text{target}}(t)$, when the ESN is driven by the training input $\mathbf{u}(t)$.

3.1. Original Training Method. Training the ESN using the original training methods is done in three steps. First, a random RNN with the echo state property is generated [4]. Second, the training sequence is run through the network once. If there are feedback connections, teacher forcing is used, meaning $\mathbf{y}(t)$ is replaced by $\mathbf{y}_{\text{target}}(t)$ when computing $\mathbf{x}(t+1)$ and $\mathbf{y}(t+1)$. After the first T_0 time steps, which are used to wash out the initial transient dynamics, the states of each input and internal node in each time step are stored in a state collection matrix, $\mathbf{M}$. Assuming tanh is used as output activation function, $\tanh^{-1}(\mathbf{y}_{\text{target}}(t))$ is collected row-wise

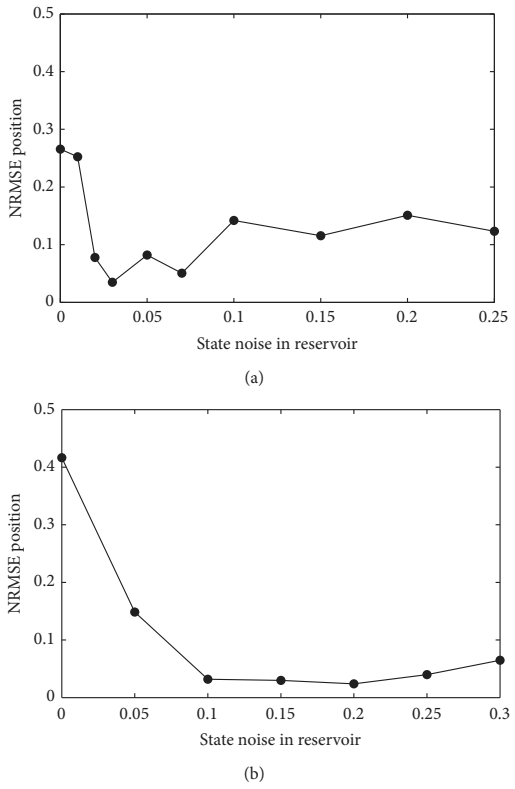


Figure 5: The optimal choice for the level of internal noise in the reservoir was significantly different for the two training methods. The figures show the mean position error during testing for different noise levels. (a) The networks were trained with the original method with the training sequence consisting of 5 repetitions of the YMCA movement. (b) The corresponding results when the networks were trained with the new method with $\beta = 0.1$ and the movement sequence repeated once. All experiments were run 10 times, and the number of internal nodes was 200 in all the networks. Based on the results we chose noise level 0.03 for the original method and 0.2 for the new method.

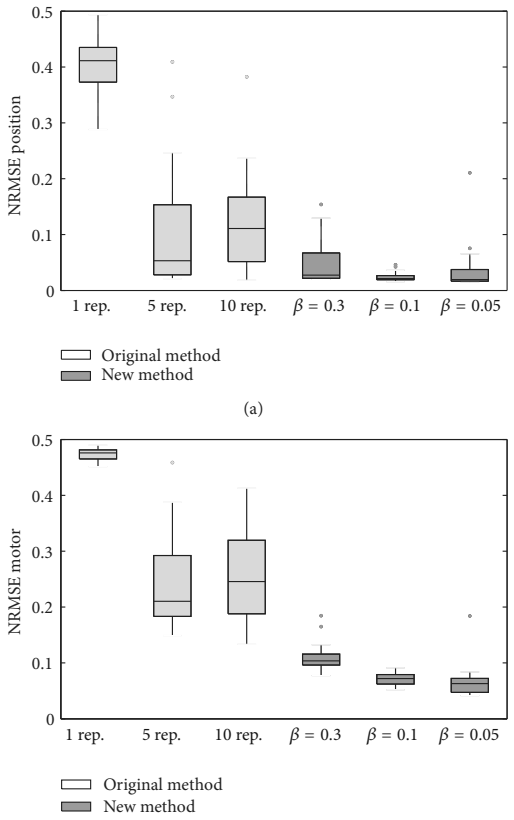


Figure 6: The figures show a box and whisker plots for 20 runs of each of the 6 experiments. Plot (a) illustrates the position error during testing and plot (b) the motor error during testing. On each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers, and outliers are plotted individually.

into a target collection matrix $\mathbf{S}$. Equation (2) can then be written as

$$\mathbf{S} = \mathbf{M}\left(\mathbf{W}^{\text{out}}\right)^{T}. \tag{3}$$

Third, the output weights are computed by using the Moore-Penrose pseudoinverse to solve (3) with regard to $\mathbf{W}^{\text{out}}$:

$$\left(\mathbf{W}^{\text{out}}\right)^{T} = \mathbf{M}^{+}\mathbf{S}. \tag{4}$$

3.2. New Proposed Training Method. In the original training method the training sequence is run through the network once, and the output weights are updated based on the target collection matrix and the state collection matrix as shown in (4). This does not work well with our training architecture, because teacher forcing and target estimation are far from perfect. We therefore suggest *running the training sequence through several times* for each value of the feedback gain. For each of these cycles the output weights are calculated based on the state collection matrix and something in between the estimated target and the actual output from the ESN model. One has

$$\mathbf{y}^{i}_{\text{used target}}(t) = \beta \mathbf{y}_{\text{estimated target}}(t) + \left(1 - \beta\right)\ \mathbf{y}_{\text{ESN}}(t). \tag{5}$$

The vector $\mathbf{y}^{i}_{\text{used target}}(t)$ is the target used to generate the target matrix $\mathbf{S}$ for computing $\mathbf{W}^{\text{out}}$ in cycle i, and $\mathbf{y}_{\text{estimated target}}(t)$ is an estimate of the target, as the true target is not available. Note that $\beta = 1$ corresponds to the original training method.

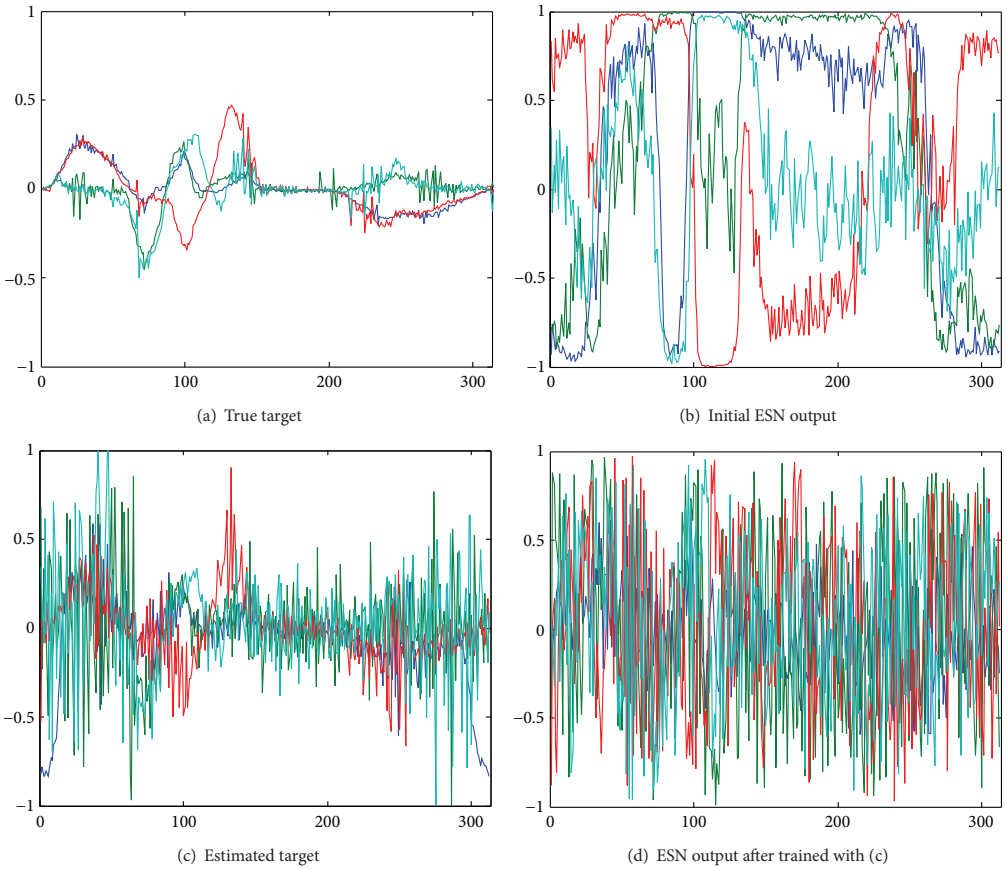


FIGURE 7: The plots illustrate why the original method without repetitions (experiment 1) fails. Compared to the true target (a), the estimated target in the first epoch (c) is very noisy. It has the general shape of the true target, but when training the initial, random ESN (b) with this noisy estimate, the result is a network which outputs mostly noise (d). This only gets worse in the succeeding epochs. Plotted are motor commands (joint angle velocities) for the 4 DOFs at each time step in the training sequence.

We hypothesize that this new proposed training method will improve learning. However, the training time increases as β decreases because additional cycles of training are needed. To test how many cycles are needed to converge for each value of β, the network was trained with the true target and perfect teacher forcing for 400 cycles. The true target was found by using an analytical inverse model. Figure 3 illustrates the difference between the true target, $\mathbf{y}_{\text{target}}$, and the used target, $\mathbf{y}_{\text{used target}}$, in each cycle, i. To compensate for this extra computation time, we will try reducing the length of the training sequence when applying this training method.

4. Experiments

The performance of the new proposed method is compared to the performance of the original method through different experiments. Our main hypothesis is that the new method will provide the same or better performance as the original at a smaller computational cost.

In all the experiments the ESN was trained to execute the YMCA movement. It was trained with feedback-error learning with the feedback gain linearly being decreased from 1 to 0 during 10 epochs of training. During testing the ESN was run without the feedback controller and the performance was measured as how accurately the ESN was able to reproduce the training sequence.

The original training method was used on training sequences with varying number of repetitions of the YMCA movement. We hypothesize that training on longer sequences, where the movement is repeated several times, will increase the performance. However, a longer training sequence leads to longer training time.

The new training method was investigated by conducting experiments for three different values of β. All trained

Table 1: The table summarizes the experiment details, including the value of β ($\beta = 1$ means the original method), the number of cycles per epoch, and the number of repetitions of the YMCA movement constituting the training sequence. In all the experiments the ESN was trained for 10 epochs with decreasing feedback gain.

Experiment number	β	# cycles per epoch	# rep. movement
Exp. 1	1	1	1
Exp. 2	1	1	5
Exp. 3	1	1	10
Exp. 4	0.3	2	1
Exp. 5	0.1	3	1
Exp. 6	0.05	10	1

on just one repetition of the YMCA movement, but the sequence had to be presented several times for each epoch to make it possible for the used target to converge during the 10 training epochs. The number of cycles used for each epoch was the approximate number of cycles needed for convergence according to Figure 3, divided by the number of epochs.

Table 1 holds the details of the different experiments.

4.1. Parameters. The ESN had 8 input nodes, corresponding to the x and z coordinates of the shoulders and elbows, and 4 output nodes, one for each DOF of the simulator. We used 200 nodes in the internal network, which was optimized for the original training method as illustrated in Figure 4.

When implementing the ESN, we used the simple matlab toolbox provided by Jaeger et al. [16]. The spectral radius was 0.5 and tanh was used as output function. The reservoir noise level was set to 0.03 when using the original method and 0.2 when using the new method. These noise levels are justified in Figure 5. All other network parameters used were the default in the toolbox. Gaussian noise with mean 0 and standard deviation 0.01 was added to the output from the arm simulator.

4.2. Training and Testing. The feedback controller was only used during training, and the feedback gain was reduced from 1 to 0 during 10 epochs. Before each epoch the ESN was reinitialized by setting the internal states to 0 and running the training sequence through once without learning. The epoch continued with one cycle of training when using the original training method and several cycles of training when $\beta < 1$. One last circle without training (but with use of the feedback controller) was run in each epoch to evaluate the performance at that stage.

After training the network was again reinitialized and tested on the training sequence by running it through once without the feedback controller.

To evaluate the performance we use the Root Mean Square Error (RMSE) of the resulting position sequence normalized over the range of the output values:

$$\mathrm{MSE}\left(\mathbf{y}, \mathbf{y}_{\text{true target}}\right) = \frac{\sqrt{\mathrm{MSE}}}{y_{\max} - y_{\min}} = \frac{\sqrt{\mathrm{MSE}}}{2}. \quad (6)$$

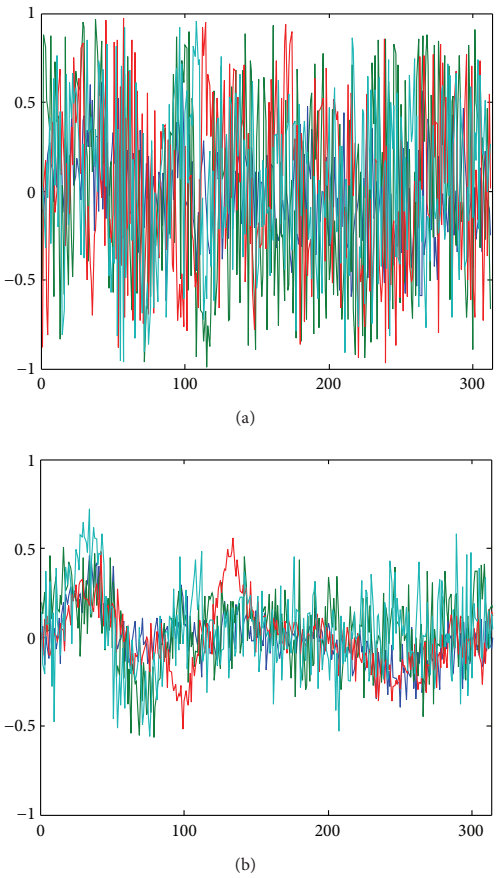


Figure 8: Adding more repetitions of the movement in the training sequence makes the output of the ESN seem less noisy. Plot (a) shows the output of the ESN after training with one repetition and plot (b) the ESN output after training on 5 repetition of the YMCA movement.

The NRMSE for each run was averaged over all time steps and DOFs. A NRMSE = 0 means no error, a random solution would have NRMSE $\approx$ 0.5, and NRMSE = 1 means opposite solution.

5. Results

Each of the six experiments were repeated 20 times, and the results are summarized in Table 2 and illustrated in Figure 6.

The motor error of experiment 1 is close to 0.5, which means that using the original training method on one repetition of the YMCA sequence results in a network that does not perform better than a random network. Repeating the movement in the training sequence (experiments 2 and 3) helps, but note that the variance is pretty large.

Using the new training method makes a larger improvement with a lower additional computational cost. From the box and whisker plot in Figure 6(b) we see that the worst ESN obtained by using the new method with $\beta = 0.1$ (experiment 5) performed better than the best ESN obtained with the original method trained on 5 repetitions of the YMCA (experiment 2). Due to the computation time of the pseudo-inverse calculations, the training time of a sequence of length $m * n$ is longer than training a sequence of length m n times [17]. This implies that the running time of experiment 5 (sequence of 313 steps run $3 * 10$ times) is also shorter than the running time of experiment 2 (sequence of $5 * 313$ steps run 10 times).

5.1. Why the New Method Outperforms the Original. To understand the effects of the different experimental setups we trained the same initial network with the setups in experiments 1 (original, 1 rep.), 2 (original, 5 rep.), and 5 (new, $\beta = 0.1$) and studied how the ESN output, the actual position sequence, the estimated target, and the target used for weight calculation evolved during the training epochs.

Figure 7 shows why experiment 1 fails. The estimated target sequence is too noisy, and with the short training sequence without any repetitions, the output from the ESN becomes even noisier.

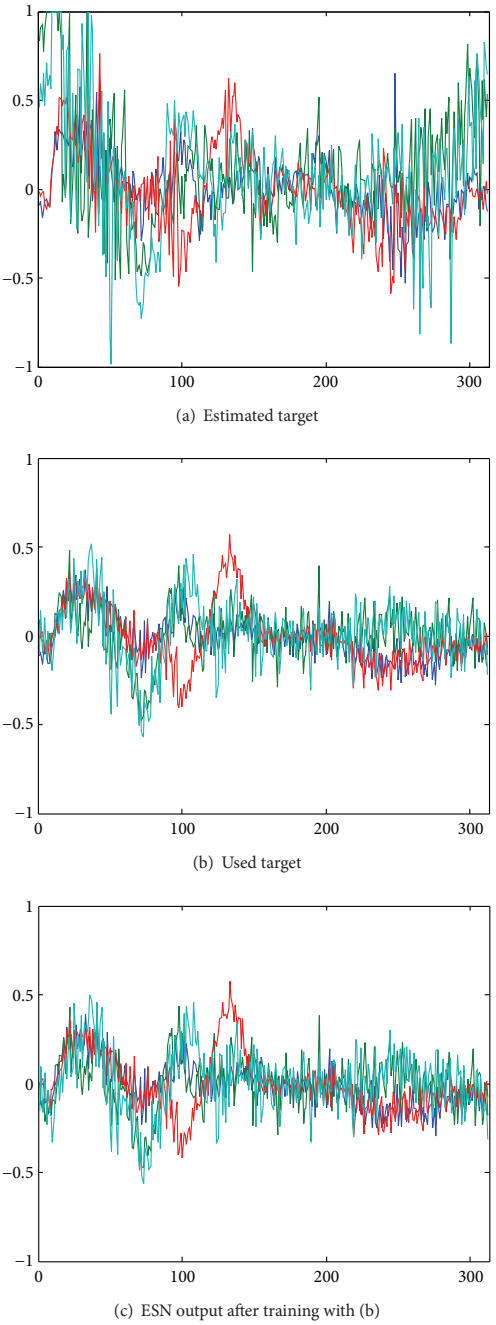
The output from the ESN after training becomes significantly less noisy when the movement is repeated several times in the training sequence, as illustrated in Figure 8. In this setup the target sequence does have a repeating pattern, and since the error in each repetition will differ, the weight calculation will average over these slightly different representations.

When using the new training method, the approach for making a smoother target is different. The new method is apparently able to keep the smoothness of the output of the first, random network and just gradually drives that solution toward the target. As illustrated in Figure 9 the used target, that is, the best target estimate combined with the previous ESN output, appears much less noisy than the target estimate alone.

The new method also results in better teacher forcing. Figure 10 illustrates the quality of the teacher forcing for the three selected experiments.

6. Discussion and Conclusion

This paper investigates using feedback-error learning to train an ESN to learn the inverse kinematics of an arm movement. When applying feedback-error learning, teacher forcing is not perfect, and joint constraints on the simulator make the feedback error inaccurate. A novel training method is suggested, which uses a combination of the previous ESN output and the estimated target to train the network. This presumably keeps much of the smoothness of the output from the initial, random network and avoids the unstable output obtained when training with the estimated target directly.



(a) Estimated target

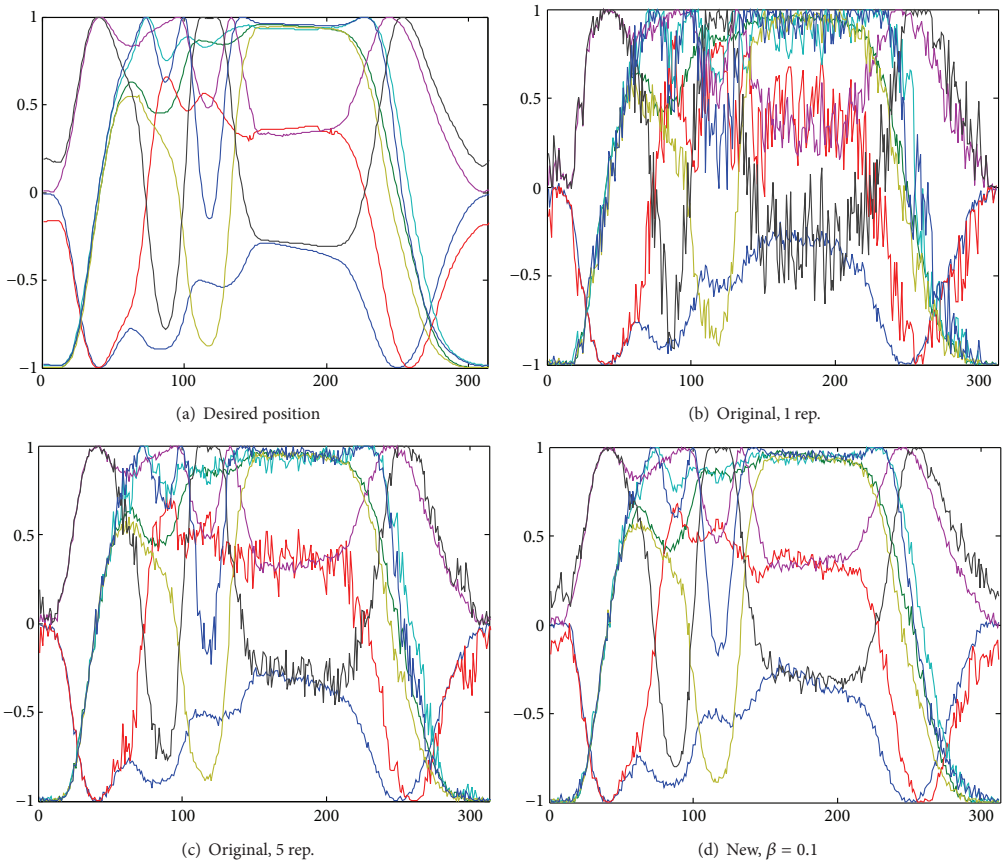
(b) Used target

(c) ESN output after training with (b)

Figure 9: In experiment 5 the network is trained on one repetition of the YMCA movement with $\beta = 0.1$. The plots show (a) the estimated target, (b) the used target, and (c) the ESN output after training with (b). All the plots are from epoch 5, where the used target is starting to look like the true target. Notice that the used target appears less noisy than the estimated target.

Table 2: The mean NRMSE and variance for 20 repetitions of each experiment. All the networks were tested on one repetition of the YMCA movement.

Experiment	Position error	Var	Motor error	Var
1 Orig. method, 1 rep.	0.4000	0.0024	0.4737	$1.4E-04$
2 Orig. method, 5 rep.	0.1088	0.0125	0.2435	0.0071
3 Orig. method, 10 rep.	0.1193	0.0081	0.2500	0.0066
4 New method, $\beta = 0.3$	0.0494	0.0018	0.1100	$6.9E-04$
5 New method, $\beta = 0.1$	0.0245	$7.0E-05$	0.0717	$1.4E-04$
6 New method, $\beta = 0.05$	0.0385	0.0020	0.0669	$9.2E-04$



(a) Desired position

(b) Original, 1 rep.

(c) Original, 5 rep.

(d) New, $\beta = 0.1$

Figure 10: The figure illustrates the quality of the teacher forcing in experiments 1, 2, and 5. For each of these experiments the position sequences in epoch 5 are plotted as the 8 coordinate values at each time step for one repetition of the YMCA movement.

The new method requires extra training cycles to converge, but we showed that this can be compensated by using a shorter training sequence.

For benchmark sequences like generation of the figure-eight [18] or a chaotic attractor like the Mackey-Glass system [19], it will be interesting to see whether this new method could be faster than the original method, as it can get the same performance by training on a shorter training sequence. Preliminary results on the generation of the figure-eight verify that a shorter training sequence is needed with the new method, but the potential computational benefits are not yet extensively tested.

References

[1] K. Doya, "Universality of fully connected recurrent neural networks," Tech. Rep., University of California, San Diego, Calif, USA, 1993, Submitted to: *IEEE Transactions on Neural Networks*.

[2] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[3] T. Natschläger, W. Maass, and H. Markram, "The "liquid computer": a novel strategy for real-time computing on time series," *Special Issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. 1, pp. 39–43, 2002.

[4] H. Jaeger, "A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach," Tech. Rep., Fraunhofer Institute for Autonomous Intelligent Systems, Sankt Augustin, Germany, 2002.

[5] J. J. Steil, "Backpropagation-decorrelation: online recurrent learning with O(N) complexity," in *Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN '04)*, pp. 843–848, July 2004.

[6] B. Schrauwen, D. Verstraeten, and J. van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proceedings of the 15th European Symposium on Artificial Neural Networks*, vol. 4, pp. 471–482, 2007.

[7] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life*, Lecture Notes in computer Science, pp. 588–597, Springer, Berlin, Germany, 2003.

[8] D. Nquyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, 2011.

[9] M. Oubbati, M. Schanz, and P. Levi, "Kinematic and dynamic adaptive control of a nonholonomic mobile robot using a RNN," in *Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA '05)*, pp. 27–33, June 2005.

[10] M. Kawato, "Feedback-error-learning neural network for supervised motor learning," in *Advanced Neural Computers*, R. Eckmiller, Ed., pp. 365–372, Elsevier, Amsterdam, The Netherlands, 1990.

[11] M. I. Jordan and D. E. Rumelhart, "Forward models: supervised learning with a distal teacher," *Cognitive Science*, vol. 16, no. 3, pp. 307–354, 1992.

[12] M. Kawato, "Internal models for motor control and trajectory planning," *Current Opinion in Neurobiology*, vol. 9, no. 6, pp. 718–727, 1999.

[13] R. A. Løvlid, "Learning to imitate YMCA with an ESN," in *Proceedings of the 22nd International Conference on Artificial Neural Networks and Machine Learning (ICANN '12)*, Lecture Notes in Computer Science, pp. 507–514, Springer, 2012.

[14] R. A. Løvlid, "Learning motor control by dancing YMCA," *IFIP Advances in Information and Communication Technology*, vol. 331, pp. 79–88, 2010.

[15] A. Tidemann and P. Öztürk, "Self-organizing multiple models for imitation: teaching a robot to dance the YMCA," in *IEA/AIE*, vol. 4570 of *Lecture Notes in Computer Science*, pp. 291–302, Springer, Berlin, Germany, 2007.

[16] H. Jaeger et al., "Simple toolbox for esns," 2009, http://reservoir-computing.org/software.

[17] F. Toutounian and A. Ataei, "A new method for computing Moore-Penrose inverse matrices," *Journal of Computational and Applied Mathematics*, vol. 228, no. 1, pp. 412–417, 2009.

[18] F. Wyffels, B. Schrauwen, and D. Stroobandt, "Stable output feedback in reservoir computing using ridge regression," in *Proceedings of the 18th International Conference on Artificial Neural Networks, Part I (ICANN '08)*, pp. 808–817, Springer, 2008.

[19] H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," Tech. Rep., GMD, 2001.

III

POSTSCRIPT

LIST OF FIGURES

BIBLIOGRAPHY

[1] Michael A. Arbib, Giorgio Metta, and Patrick van der Smagt. Neurorobotics: From vision to action. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, pages 1453–1480. Springer, 2008.

[2] Amir F. Atiya and Alexander G. Parlos. New results on recurrent network training: unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

[3] Arpan Banerjee and Viktor K. Jirsa. How do neural connectivity and time delays influence bimanual coordination? *Biological Cybernetics*, 96(2):265–278, 2007.

[4] A. Baranes and P.-Y. Oudeyer. Intrinsically motivated goal exploration for active motor learning in robots: A case study. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 1766–1773, 2010.

[5] Amy J. Bastian. Learning to predict the future: the cerebellum adapts feedforward movement control. *Current Opinion in Neurobiology*, 16:645–649, 2006.

[6] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In Bruno Siciliano and Oussama Khatib, editors, *Handbook of Robotics*, chapter 59. Springer, 2008.

[7] E. Datteri, G. Asuni, G. Teti, C. Laschi, P. Dario, and E. Guglielmelli. Experimental analysis of the conditions of applicability of a robot sensorimotor coordination scheme based on expected perception. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 2, pages 1311–1316 vol.2, 2004.

[8] Edoardo Datteri, Giancarlo Teti, Cecilia Laschi, Guglielmo Tamburrini, Paolo Dario, and Eugenio Guglielmelli. Expected perception: an aticipation-based perception-action scheme in robots. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 2003.

[9] Edoardo Datteri, Giancarlo Teti, Cecilia Laschi, Guglielmo Tamburrini, Paolo Dario, Eugenio Guglielmelli, and Scuola Superiore Sant'anna. Expected perception in robots: a biologically driven perception-action scheme. In *in Proceedings of ICAR 2003, 11th International Conference on Advanced Robotics, Vol.3*, pages 1405–1410, 2003.

[10] Pierre Dauchez, Alain Fournier, and Rene Jourdan. Hybrid control of a two-arm robot for complex tasks. *Robotics and Autonomous Systems*, 5(4):323 – 332, 1989.

[11] Yiannis Demiris and Anthony Dearden. From motor babbling to hierarchical learning by imitation: a robot developmental pathway. In *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, pages 31–37, 2005.

[12] J. Diedrichsen, R. Shadmehr, and R. B. Ivry. The coordination of movement: optimal feedback control and beyond. *Trends in Cognitive Sciences*, in press.

[13] Kenji Doya. Universality of fully connected recurrent neural networks. Technical report, University of California, San Diego, 1993. Submitted to: IEEE Transactions on Neural Networks.

[14] Kenji Doya. What are the computations of the cerebellum, the basal ganglia, and the cerebral cortex? *Neural Networks*, 12:961–974, 1999.

[15] Kenji Doya. Complementary roles of basal ganglia and cerebellum in learning and motor control. *Current opinion in neurobiology*, 10(6):732–739, 2000.

[16] Kenji Doya, Hidenori Kimura, and Mitsuo Kawato. Neural mechanisms of learning and control. *Control Systems, IEEE*, 21(4):42–54, 2001.

[17] Aaron D'Souza, Sethu Vijayakumar, and Stefan Schaal. Learning inverse kinematics. In *Proceedings of the IEEE/RSJ International Conference on Intelligence in Robotics and Autonomous Systems (IROS2001)*, 2001.

[18] Xavier Dutoit, Benjamin Schrauwen, Jan Van Campenhout, Dirk Stroobandt, Hendrik Van Brussel, and Marnix Nuttin. Pruning and regularization in reservoir computing: a first insight. In *Proceedings of the 16th European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 1–6, 2008.

[19] Herbert Jaeger et. al. Simple toolbox for ESNs. `http://reservoir-computing.org/software`, 2009.

[20] Chrisantha Fernando and Sampsa Sojakka. Pattern recognition in a bucket. In *Advances in Artificial Life*, Lecture Notes in computer Science, pages 588–597. Springer, 2003.

[21] Tamar Flash and Neville Hogans. The coordination of arm movements: An experimentally confirmed mathematical model. *Journal of neuroscience*, 5:1688–1703, 1985.

[22] Ann M. Graybiel. Building action repertoires: memory and learning functions of the basal ganglia. *Current Opinion in Neurobiology*, 5:733–741, 1995.

[23] Ann M. Graybiel. The basal ganglia and chunking of action repertoires. *Neurobiology of Learning and Memory*, 70:119–136, 1998.

[24] Ann M. Graybiel. The basal ganglia. *Current Biology*, 10(14):509–511, 2000.

[25] Elena V. Gribovskaya and Aude G. Billard. Combining dynamical systems control and programming by demonstration for teaching discrete bimanual coordination tasks to a humanoid robot. In *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2008.

[26] Horst-Michael Gross, Volker Stephan, and Torsten Seiler. Neural architecture for sensorimotor anticipation. *Cybernetics and Systems Research*, 2:593–598, 1998.

[27] Heiko Hoffman. Perception through visuomotor anticipation in a mobile robot. *Neural Networks*, 20(1):22–33, 2007.

[28] Masao Ito. Control of mental activities by internal models in the cerebellum. *Nature Reviews Neuroscience*, 9:304–313, 2008.

[29] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. Technical report, GMD, 2001.

[30] Herbert Jaeger. A tutorial on training recurrent neural networks, covering bppt, rtrl, and the echo state network approach. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems, 2002.

[31] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Sience*, 304:78–80, 2004.

[32] Michael I. Jordan. Chapter 2 computational aspects of motor control and motor learning. In Herbert Heuer and Steven W. Keele, editors, *Motor skills*, volume 2 of *Handbook of Perception and Action*, pages 71 – 120. Academic Press, 1996.

[33] Michael I. Jordan and David E. Rumelhart. Forward models: Supervised learning with a distal teacher. *Cognitive Science*, 16:307–354, 1992.

[34] M. Kawato. Cerebellum and motor control. In Michael A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks, second edition*, pages 190–195. The MIT Press, 2002.

[35] Mitsuo Kawato. Feedback-error-learning neural network for supervised motor learning. In R. Eckmiller, editor, *Advanced Neural Computers*, pages 365–372. Elsevier, 1990.

[36] Mitsuo Kawato. Internal models for motor control and trajectory planning. *Current Opinion in Neurobiology*, 9(6):718–727, 1999.

[37] Mitsuo Kawato and Hiroaki Gomi. The cerebellum and vor/okr learning models. *TINS*, 15(11), 1992.

[38] Mitsuo Kawato and Hiroaki Gomi. A computational model of four regions of the cerebellum based on feedback-error learning. *Biological Cybernetics*, 68(2):95–103, 1992.

[39] Jon Klein. breve: a 3d simulation environment for multi-agent simulations and artificial life. `http://www.spiderland.org/`.

[40] Carl D. Kopf. Dynamic two arm hybrid position/force control. *Robotics and Autonomous Systems*, 5(4):369 – 376, 1989.

[41] Cecilia Laschi, Gioel Asuni, Eugenio Guglielmelli, Giancarlo Teti, Roland Johansson, Hitoshi Konosu, Zbigniew Wasik, Maria Chiara Carrozza, and Paolo Dario. A bio-inspired predictive sensory-motor coordination scheme for robot reaching and preshaping. *Autonomous Robots*, 25(1-2):85–101, 2008.

[42] Mantas Lukoševičius. *A practical guide to applying echo state networks*, volume 7700 of *Lecture Notes in Computer Science*, pages 659–686. Springer Berlin Heidelberg, 2 edition, 2012.

[43] Mantas Lukoševičius. Echo state networks with trained feedbacks, school of engineering and science. Techreport, International University of Bremen, February 2007.

[44] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127 – 149, 2009.

[45] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[46] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[47] Stephen Marsland. *Support Vector Machines*, chapter 5, pages 119–131. Chapman & Hall, 2009.

[48] Biren Mehta and Stefan Schaal. Forward models in visuomotor control. *Journal of Neuropysiology*, 88(2):942–953, 2002.

[49] Andrew N. Meltzoff and M. Keith Moore. Explaining facial imitation: A theoretical model. *Early Development and Parenting*, 6:179–192, 1997.

[50] R.C. Miall, D.J. Weir, Daniel M. Wolpert, and J.F. Stein. Is cerebellum a smith predictor? *Journal of Motor Behavior*, 25(3):203–216, 1993.

[51] Frank A. Middleton and Peter L. Strick. Basal ganglia and cerebellar loops: motor and cognitive circuits. *Brain Research Reviews*, 31:236–250, 2000.

[52] Vishwanathan Mohan and Pietro Morasso. Passive motion paradigm: an alternative to optimal control. *Frontiers in Neurorobotics*, 5, 2011.

[53] Vishwanathan Mohan, Pietro Morasso, Giorgio Metta, and Giulio Sandini. A biomimetic, force-field based computational model for motion planning and bimanual coordination in humanoid robots. *Autonomous Robots*, 27:291–307, 2009.

[54] Pietro Morasso, Vishwanathan Mohan, Giorgio Metta, and Giulio Sandini. Motion planning and bimanual coordination in humanoid robots. In *Frontiers in Artificial Intelligence and Applications*, volume 196 of *Computational Intelligence and Bioengineering*, pages 169–185. IOS Press, 2009.

[55] Thomas Natschläger, Wolfgang Maass, and Henry Markram. The "liquid computer": A novel strategy for real-time computing on time series. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8(1):39–43, 2002.

[56] Fernando Passold and Marcelo Ricardo Stemmer. Feedback error learning neural network applied to a scara robot. In *Fourth International Workshop on Robot Motion and Control*, pages 1547–1554. MIT Press, 2004.

[57] B.A. Pearlmutter. Gradient calculations for dynamic recurrent neural networks: a survey. *Neural Networks, IEEE Transactions on*, 6(5):1212–1228, 1995.

[58] Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[59] Tony J. Prescott, Fernando M. Montes González, Kevin Gurney, Mark D. Humphries, and Peter Redgrave. A robot model of the basal ganglia: Behavior and intrinsic processing. *Neural Networks*, 19(1):31 – 61, 2006.

[60] R.F. Reinhart and J.J. Steil. Recurrent neural associative learning of forward and inverse kinematics for movement generation of the redundant pa-10 robot. In *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS '08. ECSIS Symposium on*, pages 35–40, 2008.

[61] Matthias Rolf, Jochen J. Steil, and Michael Gienger. Efficient exploration and learning of whole body kinematics. In *IEEE 8th International Conference on Development and Learning*, 2009.

[62] S. Schaal and C.G. Atkeson. Robot juggling: implementation of memory-based learning. *Control Systems, IEEE*, 14(1):57–71, 1994.

[63] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pages 471–482, 4 2007.

[64] Carol A. Seger. The basal ganglia in human learning. *The Neuroscientist*, 12(2), 2006.

[65] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison, 2009.

[66] R. Shadmehr and S.P. Wise. *The Computational Neurobiology of Reaching and Pointing: A Foundation for Motor Learning*. MIT Press, 2005.

[67] Reza Shadmehr and John W. Krakauer. A computational neuroanatomy for motor control. *Experimental Brain Research*, 185(3):359–381, 2008.

[68] Qingsong Song, Zuren Feng, and Mingli Lei. Stable training method for echo state networks with output feedbacks. In *Networking, Sensing and Control (ICNSC), 2010 International Conference on*, pages 159 –164, april 2010.

[69] Jochen J. Steil. Backpropagation-decorrelation: online recurrent learning with o(n) complexity. In *Proc. IJCNN*, 2004.

[70] Stephan P. Swinnen. Intermanual coordination: From behavioural principles to neural-network interactions. *Nature*, 3:348–359, 2002.

[71] Axel Tidemann and Pinar Öztürk. Self-organizing multiple models for imitation: Teaching a robot to dance the YMCA. In *IEA/AIE*, volume 4570 of *LNCS*, pages 291–302. Springer, June 2007.

[72] Deepak Tolani and Norman I. Badler. Real-time inverse kinematics of the human arm. *Presence*, 5(4):393–401, 1996.

[73] Y. Uno, M. Kawato, and R. Suzuki. Formation and control of optimal trajectory in human multijoint arm movement. *Biological Cybernetics*, 61(2):89–101, 1989.

[74] Patrick van der Smagt. Cerebellar control of robot arms. *Connection Science*, 10:301–320, 1998.

[75] Patrick van der Smagt and Daniel Bullock. Extended abstracts of the nips*97 workshop - can artificial cerebellar models compete to control robots? Technical report, DLR.

[76] Patrick van der Smagt and Gerd Hirzinger. The cerebellum as computed torque model. In R.J. Howlett and L.C. Jain, editors, *Fourth International Conference on Knowledge-Based Ingelligent Engineering Systems & Applied Technologies*, 2000.

[77] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20:391–403, 2007.

[78] Sethu Vijayakumar and Stefan Schaal. Locally weighted projection regression: An $o(n)$ algorithm for incremental real time learning in high dimensional spaces. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000)*, 2000.

[79] Claes von Hofsen. An action perspective on motor development. *TRENDS in Cognitive Sciences*, 8(6):266–272, 2004.

[80] Daniel M. Wolpert, Zoubin Ghahramani, and Michael I. Jordan. An internal model for sensorimotor integration. *Science*, 269:1880–1882, 1995.

[81] Daniel M. Wolpert and Mitsuo Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11:1317–1329, 1998.

[82] Daniel M. Wolpert, R. Chris Miall, and Mitsuo Kawato. Internal models in the cerebellum. *Trends in Cognitive Sciences*, 2(9), 1998.

[83] David H. Wolpert. The supervised learning no-free-lunch theorems. In *Proceedings of the 6th Online World conference on Soft Computing in Industrial Applications (WSC 2006)*, pages 25–42, 2006.

[84] Francis Wyffels, Benjamin Schrauwen, and Dirk Stroobandt. Stable output feedback in reservoir computing using ridge regression. In *Proceedings of the 18th international conference on Artificial Neural Networks, Part I*, ICANN '08, pages 808–817, Berlin, Heidelberg, 2008. Springer-Verlag.

[85] Tadashi Yamazaki and Shigeru Tanaka. The cerebellum as a liquid state machine. *Neural Networks*, 20:290–297, 2007.