Adaptieve controle van soepele robots met Reservoir Computing

Adaptive Control of Compliant Robots with Reservoir Computing

Tim Waegeman

Promotor: prof. dr. ir. B. Schrauwen
Proefschrift ingediend tot het behalen van de graad van
Doctor in de Ingenieurswetenschappen: Computerwetenschappen

Vakgroep Elektronica en Informatiesystemen
Voorzitter: prof. dr. ir. J. Van Campenhout
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2013 - 2014

UNIVERSITEIT
GENT

ISBN 978-90-8578-652-8
NUR 984
Wettelijk depot: D/2013/10.500/85

Dankwoord

Als ingenieurstudent had ik het genoegen om zowel mijn bachelorproef als mijn masterproef bij het Reservoir Lab te kunnen uitvoeren. Nadat ze mij met open armen hadden verwelkomd als doctoraatstudent, begon voor mij een uitdagende en interessante periode van vier jaar waarin ik veel heb bijgeleerd. Het doen van een doctoraat geeft je de kans om je op zelfstandige basis verder te ontwikkelen in een wetenschappelijke en stimulerende omgeving. Dit alles zou natuurlijk niet mogelijk geweest zijn zonder de medewerking van een aantal mensen. Ik wil dit dankwoord dan ook gebruiken om hen uitvoerig te bedanken.

In de eerste plaats wil ik mijn promotor prof. Benjamin Schrauwen bedanken voor zijn vele ideeën, kritische blik en aanstekelijk enthousiasme. Ik herinner mij nog beeldig de eerste maanden van mijn doctoraat waarin Ben bijna dagelijks een nieuw stapeltje papers kwam aandragen met de bijbehorende ideeën. Naast mijn promotor wil ik ook prof. Joni Dambre, prof. Jan Van Campenhout, prof. Gert De Cooman, prof. Bram Vanderborght, prof. Fumiya Iida en prof. Patrick Van Der Smagt bedanken voor het grondig en met een kritische blik lezen van mijn thesis, ter verhoging van het taalkundig en wetenschappelijk niveau ervan.

I wish to express my gratitude to the members of my jury for giving me valuable advice and lifting the text up to its current level.

Het Reservoir Lab heeft zijn stimulerende omgeving natuurlijk ook te danken aan alle andere collega's. Francis, bedankt voor het leuke gezelschap, de vele succesvolle vormen van samenwerking, binnen en buiten het lab, en voor de motiverende begeleiding gedurende mijn bachelor- en masterproef. Mijn kantoorgenoten mogen hier natuurlijk ook niet ontbreken. Pieter en Michiel H. bedankt voor de leuke sfeer en jullie oprechte vriendschap die mij de kans heeft gegeven om jullie vrienden ook mijn vrienden te kunnen noemen. Philémon, jij ook bedankt voor de

leuke, al dan niet werkgerelateerde, gesprekken en je vaste paraatheid als ik eens een stapje wilde zetten. De rest van mijn collega's wil ik hierbij ook bedanken: David V., Michiel DH., Eric, Fiontann en Antonio, die ondertussen ons lab hebben verlaten; Pieter-Jan: die ook buiten het lab voor de nodige activiteit zorgde; Sander en Aäron: met wie ik leuke computer/gezelschapspelletjes heb gespeeld; Thibault; Jonas, Michael en Ken: met wie ik vaak de robots gezelschap heb gehouden; Ken: ook bedankt voor de nodige afleiding buiten het lab. *Juan Pablo, our latest valuable addition with which I had many insightful and pleasant discussions*; Marnix, bedankt voor je vaste paraatheid bij het oplossen van SAP-problemen.

I would like to take this opportunity to thank the people of the LASA-lab at EPFL for hosting me for one month, giving me access to their robots and making me feel being a part of their lab.

De naaste vrienden, die voor de nodige afleiding en ontspanning zorgden, mogen hier natuurlijk ook niet ontbreken: Wim D., Laure, Karen, Paul, Guillaume, Sven, Daan, Mattias, Michiel S., Raf, Stan, Jens, Willem, Elias, Tom, Brahim, Karel B., Karel H., Peter, Koen, Wim DM., Tim B., Jonas, David H.; mijn scoutsvrienden: Jochem, Jeroen, Wim K.; mijn huisgenoten: Bieke, Anne-Loes, Sofie[2], Tim D. en Thomas; mijn surf-, klim-, zwem- en waterpolovrienden en iedereen die ik nog vergeten ben.

Last but not least: Mama en Papa, bedankt voor de kansen en de ondersteuning die jullie mij altijd hebben gegeven, zonder jullie zou ik hier nooit geraakt zijn. Mijn beste vriend en broer Bart, zijn vriendin Zincke en de hele familie, bedankt!

Tim Waegeman

Dit werk is ondersteund door het Instituut voor de Aanmoediging van Innovatie door Wetenschap en Technologie Vlaanderen (IWT Vlaanderen).

This work was supported by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen).

Examencommissie

Prof. Jan Van Campenhout, voorzitter
Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Prof. Joni Dambre, secretaris
Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Prof. Benjamin Schrauwen, promotor
Vakgroep ELIS, Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Prof. Gert De Cooman
Vakgroep EESA, Faculteit Ingenieurswetenschappen en Architectuur
Universiteit Gent

Prof. Bram Vanderborght
Department of Mechanical Engineering, Faculty of Applied Sciences
Vrije Universiteit Brussel

Prof. Fumiya Iida
Bio-Inspired Robotics Lab, Institute of Robotics and Intelligent Systems
Eidgenössische Technische Hochschule Zürich

Prof. Patrick Van Der Smagt
Biomimetic Robotics and Machine Learning Lab, Institute of Informatics
Technische Universität München

Samenvatting

In onze samenleving worden robots steeds meer gebruikt voor het uitvoeren van gevaarlijke, repetitieve en/of zware taken, waarbij bovendien een hoge precisie is vereist. Vanwege deze vereisten worden dergelijke robots vaak opgebouwd uit krachtige motoren en stevige materialen, waardoor ze gevaarlijk kunnen zijn voor de mens. In een autofabriek bijvoorbeeld wordt een grote kooi geplaatst rondom de werkruimte van de robot, waardoor mensen niet in de omgeving van de robot kunnen komen. Gedurende enkele decennia bestaat een toenemende belangstelling om de interactie tussen mens en robot te verbeteren. De bewegingen van robots worden vaak gezien als niet-vloeiende bewegingen waarin duidelijk deelbewegingen zijn te onderscheiden. Dit maakt acties van robots moeilijk voorspelbaar voor mensen. Daarom is de behoefte aanwezig om de bewegingsgeneratie van robots te verbeteren, met als doel een betere mens-robotinteractie te verkrijgen. Een interessant onderzoeksdomein in dit verband is het leren door imitatie, hierbij worden bewegingen opgenomen bij mensen en getoond aan de robot. Hoewel de robot in staat is om deze bewegingen te reproduceren, kan hij ze echter niet veralgemenen naar andere situaties. Hiervoor is een oplossing voorgesteld, waarbij gebruik wordt gemaakt van een dynamisch systeem om de opgenomen bewegingen in onder te brengen. Door het modificeren van de niet-lineaire dynamica van een dergelijk systeem, wordt het mogelijk om ook bewegingen te genereren voor andere ongeziene situaties.

In dit onderzoek is een Reservoir Computing (RC)-techniek gebruikt om een dynamisch systeem te creëren, waarin zulke voorbeeldbewegingen kunnen worden ondergebracht. Reservoir Computing-systemen zijn methoden gebaseerd op Recurrente Neurale Netwerken, die op een efficiënte manier worden getraind. Hierbij wordt alleen de training van de uitleesgewichten in beschouwing genomen, terwijl de andere connectiegewichten niet veranderen ten opzichte van hun willekeurig geïnitialiseerde waarden. Hoewel dergelijke RC-systemen reeds werden gebruikt om ritmische patronen te genereren, zijn ze in dit verband uitgebreid voor het genereren

van discrete bewegingen of beide. Dit proefschrift beschrijft hoe een dergelijk patroon genererend systeem wordt gebouwd, bestudeert de onderliggende dynamica en evalueert de robuustheid tijdens verstoringen. Daarnaast wordt een dynamische systeembenadering van objectontwijking gepresenteerd, die onder meer is geïnspireerd door potentiaalstromen rond een sferisch object. Deze techniek kan worden gebruikt om de bewegingsmogelijkheden van een robot uit te breiden zonder wijzigingen aan te brengen aan de getrainde patroongenerator. Hierdoor kan deze methode worden toegepast op elk systeem dat een bewegingstraject genereert, waarbij een zeer snelle objectontwijking is vereist.

Veronderstel nu dat het patroongenererend systeem wordt toegepast op een industriële robotarm, vergelijkbaar met deze die wordt gebruikt in een autofabriek: hoewel de voorgestelde objectontwijkingsstrategie in staat is om botsingen met de grijper te vermijden, is het nog steeds mogelijk dat botsingen voorkomen met de andere onderdelen van de robot. Om dit te voorkomen, zijn ingenieurs overgeschakeld op geavanceerde controletechnieken, die gebruikmaken van krachtsensoren om een indicatie te krijgen hoeveel koppel (maat voor het rotatie-effect van een kracht) op elk gewricht wordt uitgeoefend. Dankzij deze krachtmetingen is de robot in staat volgzaam te reageren op externe verstoringen. Het blijkt echter dat, zelfs met snelle regelkringen, de nodige compensaties voor een plotse storing te traag zijn om grote interactiekrachten te voorkomen. Om deze interactiekrachten te verminderen, maken onderzoekers recent gebruik van flexibele elementen (zoals veren) en lichte elastische materialen voor het construeren van robots. Hoewel dergelijke robots veel veiliger zijn, is het niet evident om hun dynamische eigenschappen te bepalen, waardoor de controle van deze robots zeer moeilijk wordt. De meeste controletechnieken maken gebruik van modelinformatie (bijvoorbeeld: gewichtsdistributie en vorm), wat niet eenvoudig is te bepalen bij dit type robots. Daarom wordt in dit proefschrift een controletechniek voorgesteld, die geen voorkennis of modelinformatie van de robot nodig heeft. Door het interageren met de robot zelf, leert de voorgestelde controller een invers robotmodel dat vervolgens kan worden gebruikt voor controle. Hoe meer de controller interageert met de robot, hoe beter de controle wordt. Hierdoor heeft deze techniek de volgende toepasselijke naam gekregen: Inverse Modeling Adaptive (IMA)-controller. Ik heb deze IMA-controller op een groot aantal taken geëvalueerd, waardoor de modelonafhankelijkheid en stabiliteit konden worden onderzocht. Verder is de snelle leercapaciteit van de IMA-controller aangetoond, tezamen met zijn vergelijkbare prestatie ten opzichte van speciaal ontworpen en taakgebonden controllers.

Wanneer zowel de voorgestelde patroongenerator en de IMA-controller worden gebruikt, wordt het mogelijk om deze moeilijk controleerbare robots te controleren in een mensvriendelijke omgeving. Wanneer het wenselijk wordt geacht dat dergelijke robots menselijke bewegingen genereren voor een groot aantal taken, dan zouden in principe voor alle mogelijke taken voorbeeldbewegingen moeten wor-

den opgenomen. Biologisch onderzoek naar de bewegingsgeneratie bij mensen en dieren heeft echter aangetoond dat een beperkte set van basisbewegingen, genaamd bewegingsprimitieven, worden gemodificeerd en gecombineerd om uiteindelijk alle mogelijke bewegingen te genereren. In dit onderzoek is verder gewerkt met deze interessante bevindingen, door te onderzoeken of één enkel bewegingsprimitief inderdaad kan worden gemodificeerd, zodat gewenste bewegingseigenschappen kunnen worden verkregen. Door enkele basisexperimenten uit te voeren waarbij een patroongenerator wordt gecontroleerd door een IMA-controller, wordt de haalbaarheid van dit concept gepresenteerd. Verder wordt een generieke controlehiërarchie geïntroduceerd, die beschrijft hoe een robot op een biologisch geïnspireerde manier kan worden gecontroleerd. Bovendien is onderzocht op welke manier bewegingsprimitieven kunnen worden gecombineerd om tot een gewenst gedrag te komen. Binnen de beperkte tijd dat ik dit aspect onderzocht heb ben ik er echter niet in geslaagd om meer geavanceerdere implementaties werkend te krijgen. In de appendix van dit proefschrift zijn wel de resultaten weergegeven van een aantal basisexperimenten.

Een ander denkpiste die ik onderzocht heb, benadert de biologische bevindingen van de bewegingsgeneratie op een andere manier. Hierbij wordt ervan uitgegaan dat de bewegingsprimitieven zelf ongedefinieerd zijn. In plaats daarvan wordt alleen een beschrijving op hoog niveau gegeven. Die beschrijft dat elke bewegingsprimitief, gemiddeld gezien, evenveel moet bijdragen aan de beweging, terwijl toch nog ruimte wordt gelaten voor specialisatie in een deel van de beweging. Zonder het gedrag van een primitief te bepalen, wordt slechts een aantal ongetrainde IMA-controllers gebruikt, die uiteindelijk de primitieven moeten voorstellen. Door deze heuristische beschrijving wordt de taakruimte van de robot op een ongesuperviseerde (zonder voorbeelden) manier opgedeeld in deelregio's. Deze Modulaire Architectuur met Controleprimitieven (MACOP) is toegepast op een robotarm waarvan de inverse kinematica moet worden geleerd. Dankzij deze opdeling van de taakruimte wordt het mogelijk om met redundante oplossingen om te gaan. Binnen één bepaalde deelregio van de taakruimte is één enkele bewegingsprimitief nauwkeuriger dan in de andere regio's. Hierdoor wordt de complexiteit van de taak opgedeeld in eenvoudiger deeltaken.

Ten slotte is het gebruik van de IMA-controller uitgebreid naar ondergeactueerde systemen. Door het gebruik van een bemonstering-gebaseerde planningstechniek wordt het mogelijk om de taak dynamica te verkennen en een pad te plannen naar de oplossing. Vervolgens wordt MACOP gebruikt om terugkoppeling te introduceren en om de bijhorende controlesignalen te leren op basis van dit geplande pad. Deze techniek blijkt bestand te zijn tegen onnauwkeurigheden in het geplande pad. Middels deze techniek wordt de geïnverteerde pendel-taak opgelost, bovendien wordt een concept van een simulatie-gebaseerd raamwerk beschreven dat in staat is om zowel de dynamica als de controle van de taak simultaan te leren.

Summary

In modern society, robots are increasingly used to handle dangerous, repetitive and/or heavy tasks with high precision. Because of the nature of the tasks, either being dangerous, high precision or simply repetitive, robots are usually constructed with high torque motors and sturdy materials, that makes them dangerous for humans to handle. In a car-manufacturing company, for example, a large cage is placed around the robot's workspace that prevents humans from entering its vicinity. In the last few decades, efforts have been made to improve human-robot interaction. Often the movement of robots is characterized as not being smooth and clearly dividable into sub-movements. This makes their movement rather unpredictable for humans. So, there exists an opportunity to improve the motion generation of robots to enhance human-robot interaction. One interesting research direction is that of imitation learning. Here, human motions are recorded and demonstrated to the robot. Although the robot is able to reproduce such movements, it cannot be generalized to other situations. Therefore, a dynamical system approach is proposed where the recorded motions are embedded into the dynamics of the system. Shaping these nonlinear dynamics, according to recorded motions, allows for dynamical system to generalize beyond demonstration. As a result, the robot can generate motions of other situations not included in the recorded human demonstrations.

In this dissertation, a Reservoir Computing approach is used to create a dynamical system in which such demonstrations are embedded. Reservoir Computing systems are Recurrent Neural Network-based approaches that are efficiently trained by considering only the training of the readout connections and retaining all other connections of such a network unchanged given their initial randomly chosen values. Although they have been used to embed periodic motions before, they were extended to embed discrete motions, or both. This work describes how such a motion pattern-generating system is built, investigates the nature of the underlying dynamics and evaluates their robustness in the face of perturbations. Additionally, a dynamical system approach to obstacle avoidance is proposed that is based on

vector fields in the presence of repellers. This technique can be used to extend the motion abilities of the robot without need for changing the trained Motion Pattern Generator (MPG). Therefore, this approach can be applied in real-time on any system that generates a certain movement trajectory.

Assume that the MPG system is implemented on an industrial robotic arm, similar to the ones used in a car factory. Even though the obstacle avoidance strategy presented is able to modify the generated motion of the robot's gripper in such a way that it avoids obstacles, it does not guarantee that other parts of the robot cannot collide with a human. To prevent this, engineers have started to use advanced control algorithms that measure the amount of torque that is applied on the robot. This allows the robot to be aware of external perturbations. However, it turns out that, even with fast control loops, the adaptation to compensate for a sudden perturbation, is too slow to prevent high interaction forces. To reduce such forces, researchers started to use mechanical elements that are passively compliant (e.g., springs) and light-weight flexible materials to construct robots. Although such compliant robots are much safer and inherently energy efficient to use, their control becomes much harder. Most control approaches use model information about the robot (e.g., weight distribution and shape). However, when constructing a compliant robot it is hard to determine the dynamics of these materials. Therefore, a model-free adaptive control framework is proposed that assumes no prior knowledge about the robot. By interacting with the robot it learns an inverse robot model that is used as controller. The more it interacts, the better the control becomes. Appropriately, this framework is called Inverse Modeling Adaptive (IMA) control framework. I have evaluated the IMA controller's tracking ability on several tasks, investigating its model independence and stability. Furthermore, I have shown its fast learning ability and comparable performance to taskspecific designed controllers.

Given both the MPG and IMA controllers, it is possible to improve the interactability of a compliant robot in a human-friendly environment. When the robot is to perform human-like motions for a large set of tasks, we need to demonstrate motion examples of all these tasks. However, biological research concerning the motion generation of animals and humans revealed that a limited set of motion patterns, called motion primitives, are modulated and combined to generate advanced motor/motion skills that humans and animals exhibit. Inspired by these interesting findings, I investigate if a single motion primitive indeed can be modulated to achieve a desired motion behavior. By some elementary experiments, where an MPG is controlled by an IMA controller, a proof of concept is presented. Furthermore, a general hierarchy is introduced that describes how a robot can be controlled in a biology-inspired manner. I also investigated how motion primitives can be combined to produce a desired motion. However, I was unable to get more advanced implementations to work. The results of some simple experiments are

presented in the appendix.

Another approach I investigated assumes that the primitives themselves are undefined. Instead, only a high-level description is given, which describes that every primitive on average should contribute equally, while still allowing for a single primitive to specialize in a part of the motion generation. Without defining the behavior of a primitive, only a set of untrained IMA controllers is used of which each will represent a single primitive. As a result of the high-level heuristic description, the task space is tiled into sub-regions in an unsupervised manner. Resulting in controllers that indeed represent a part of the motion generation. I have applied this Modular Architecture with Control Primitives (MACOP) on an inverse kinematic learning task and investigated the emerged primitives. Thanks to the tiling of the task space, it becomes possible to control redundant systems, because redundant solutions can be spread over several control primitives. Within each sub-region of the task space, a specific control primitive is more accurate than in other regions allowing for the task complexity to be distributed over several less complex tasks.

Finally, I extend the use of an IMA-controller, which is tracking controller, to the control of under-actuated systems. By using a sample-based planning algorithm it becomes possible to explore the system dynamics in which a path to a desired state can be planned. Afterwards, MACOP is used to incorporate feedback and to learn the necessary control commands corresponding to the planned state space trajectory, even if it contains errors. As a result, the under-actuated control of a cart pole system was achieved. Furthermore, I presented the concept of a simulation-based control framework that allows the learning of the system dynamics, planning and feedback control iteratively and simultaneously.

Glossary

Compliant robot A robot that is able to perform its designated tasks while compensating for unknown perturbations. The degree of compliance can range between fully compliant and fully stiff (not included). For every level of compliance there exists two types of compliance: Active compliance and Passive compliance

Control primitive A primitive is a building block that is combined, by an appropriate transformation, with a limited number of other primitives to generate a wide variety of movements. A control primitive is the control signals produced by a single controller within MACOP.

Dynamical system Is a system that evolves over time and is described by a fixed formulation, called a differential equation.

Neuron state A neuron is fully described by its activation function and when this activation function depends on time, the value of such activation function defines the neuron state. This is similar to a system state.

Network state The state of a network is fully described by all neuron states and the connections between them. When the network connections are fixed over time, the network state is fully defined by all neuron states.

System state Is a set of variables that fully describes the state of a dynamical system. These variables are used by a differential equation to describe how a dynamical system evolves.

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BIBO	Bounded Input Bounded Output
CNS	Central Nervous Systems
CPG	Central Pattern Generator
DMP	Dynamic Movement Primitive
DOF	Degrees Of Freedom
DS	Dynamical System
ESN	Echo State Network
ESP	Echo State Property
FORCE	First-Order Reduced and Controlled Error
GMR	Gaussian Mixture Regression
GP	Gaussian Process
GPR	Gaussian Process Regression
IK	Inverse Kinematics
IMA	Inverse Modeling Adaptive (control)
LMC	Linear Memory Capacity
LQR	Linear Quadratic Regulator
MACOP	Modular Architecture with Control Primitives
ML	Machine Learning
MLP	Multi Layer Perceptron
MPG	Motion Pattern Generator
MSE	Mean Square Error
NMSE	Normalized Mean Square Error
NN	Neural Network
NRMSE	Normalized Root Mean Square Error

PID	Proportional-Integral-Derivative (control)
RC	Reservoir Computing
RG	Reachability Guided
RL	Reinforcement Learning
RLS	Recursive Least Squares
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RRT	Rapidly-exploring Random Tree
SEA	Series Elastic Actuator
SVR	Support Vector Regression
WAM	Whole Arm Manipulator

List of Symbols and Notations

Δt	Time step size or sample period
$\mathbf{y}(t)$	Observation
$\mathbf{x}(t)$	Action
$\widehat{\cdot}$	Measured value
N	Number of neurons
γ	Leak rate
ρ	Spectral radius
ν	Regularization parameter
f_b^r	Input bias scaling factor
f_i^r	Input scaling factor
f_o^r	Output feedback scaling factor
$\mathbf{u}(k)$	Network input
$\mathbf{a}(k)$	Network state
$\mathbf{o}(k)$	Network output
$\mathbf{e}(k)$	Error
$\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$	Input weights of an RNN
$\mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$	Bias weights of an RNN
$\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$	Recurrent weights of an RNN
$\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$	Ouput weights of an RNN
$\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$	Output feedback weights of an RNN

$\mathbf{\Phi}$	Design matrix: concatenation of feature vectors
Υ_c	Cartesian reference frame
Υ_s	Spherical reference frame
K_P	Proportional scaling factor PID
K_I	Integral scaling factor PID
K_D	Derivative scaling factor PID
δ	Time delay IMA controller
N_c	Number of controllers (MACOP)
$\mathbf{V}(t)$	Projection matrix (MACOP)
ζ_i	Responsibility or scaling factor (MACOP)
η_g	Learning rate of growing property (MACOP)
η_s	Learning rate of scaling property (MACOP)
$\mathbf{P}(k)$	Running estimate of the inverted covariance matrix (RLS)
λ	Forgetting factor (RLS)
α	Learning rate (RLS)
$\|\cdot\|_F$	Frobenius norm
$\mathbf{I}$	Identity matrix

Contents

1 Introduction

This PhD thesis investigates the design and evaluation of an adaptive control approach for compliant robots. The underlying mechanism of this controller uses a Recurrent Neural Network that is trained by a Reservoir Computing technique. Although it is applicable to a wide variety of tasks, the motivation of the research presented in this dissertation is the need for new control approaches that can control robots that are hard or even impossible to model. After elaborating on the use of robots, and compliant robots in particular, this chapter will give an overview of some of the properties of compliant robots in which the motivation of this PhD thesis is situated. Afterwards, I will introduce the Machine Learning context in which the successive chapters are written. Finally, this chapter concludes with an overview of the dissertation's structure, the main research contributions and a publication list.

1.1 Why Robots?

Although most people have an idea about the definition of a robot, there probably does not exist a single definition on which everyone can agree. Most people associate the word robot with a human-like machine, called humanoid, which moves around exhibiting intelligent behavior. However, some of these robots are remotely controlled by humans, explaining the source of their intelligence. On the other hand there exist machines that do not resemble humans, but perform complex automated tasks, e.g., an elevator that minimizes the waiting time by planning which floor to visit first. So before discussing the underlying motivation for building and using robots in the first place, I briefly introduce a general definition of *a robot* and what it actually means. The word "robot" originates from the Czech word "robota" which means corvée or unpaid labor and was first used in a 1920's Czech play, called

Rossum's Universal Robots, to denote artificial people who can think for themselves and are eager to serve (Zunt, 2002). Given this initial interpretation, I define a robot as an automated device that can perform a wide variety of tasks and/or a task autonomously. Within this definition the following objects are robots: a not necessarily autonomously controlled manipulator that performs multiple tasks, or an autonomous device that performs a single task, e.g., a robot that cleans your floor. However, devices that perform a single task and are not fully autonomously controlled, e.g., an airplane or a remote-controlled car, are not defined as being a robot.

In this dissertation the word *plant* will be used generally to define all devices on which actions can be performed and for which the corresponding response can be observed, including robots.

Throughout history, the use of robots has become increasingly important in our society. They reduce the production time of goods, improve the accuracy of certain tasks, perform dangerous manipulations and save lives. Robots have become useful tools in our quest to discover and go beyond new frontiers. For example, the Mars rover Curiosity which successfully landed in the Gale Crater on August 6, 2012 helps us gain new insights in Mars's climate and geology. This evolution will continue to approach today's seemingly impossible frontiers while we have caught up with some science fiction stories of a few decades ago. Traces of the existence of machines which we refer to as "robot" can be found in many ancient mythologies. For instance the mechanical helper built by the Greek god Hephaestus or the statue of Pygmalion that came to life. Also in ancient China traces can be found of human-shaped mechanical devices or artificial animals such as the wooden birds that could successfully fly (Needham, 1976). We might never known the true motivations that drove humans to design and build robots. Often it is argued that man's greatest ambition is to create a living being from the ground up, in order to validate his understanding of life itself. However, another possible motivation is related to evolution. Every biological system tries to minimize its energy consumption on non-instantly-rewarding tasks in order to increase its chances to survive. For instance, humans have learned to use tools to reduce the tedious time to access some food sources. One can argue that although we have dramatically affected the evolutionary process, we still design new devices which allow us to make life more "easy" so that we can concentrate our energy on more interesting and challenging endeavors instead of washing dishes.

According to Siciliano and Khatib (2008), the field of robotics was defined in the 1980's as the science that studies the intelligent connection between perception and action. The perception is represented by the information provided by a set of sensors while the action results from a set of actuators into the robot's *locomotion* or *manipulation*. A control architecture, which encapsulates the planning and control of a robot based on the available information (e.g., sensor informa-

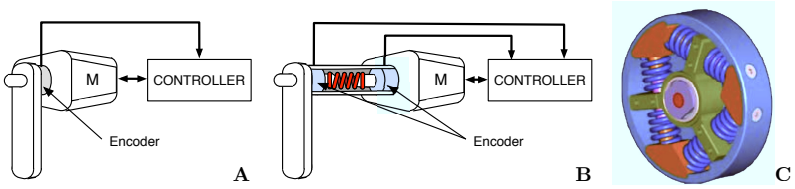


Figure 1.1: **A** illustrates a bar attached to a motor with an encoder (gray disc). **B** illustrates a series elastic actuator attached to a bar. The red spiral represents a spring. In this setup two encoders are used: one to measure the rotation induced by the passive element (spring) and one to measure the actual rotation of the motor. **C** shows a compact serial elastic actuator design used in the COMAN humanoid platform.

tion, model information, ...), defines the before mentioned intelligent connection. During the 1990's, research in the field of robotics expanded with for example the introduction of *field robotics* to fulfill a new desire to explore hazardous environments. At the same time the idea of having a robot in one's home to do some chores became more attractive. During the last decade, there has been a clear transition from having a large and heavy robot in a factory to smaller robots that can reside in an environment occupied by humans. Robots within a factory environment are built to perform repetitive and heavy tasks with high accuracy and speed. This is achieved by having high control gains that ensure the desired position of the actuators. Furthermore, these robots operate in a fenced deterministic environment where the position of every object that needs to be manipulated is known. Allowing such robots to work in an uncertain environment would cause very dangerous situations in which the presence of humans should be prevented. The transition to an uncertain environment has led researchers to investigate a new kind of robots called *Compliant Robots*.

1.2 Compliant Robots

When humans and/or animals interact with each other, they take the movements of each other into account. For instance, when we shake hands, we sense the forces applied by our counterpart and jointly comply to each other's movement. When we caress a dog, we follow the shape of the body in a manner that the applied forces are as gentle as possible. When we write a letter, we ensure that the pen point is resting on the paper, and we guide our movements according to the plane of the

table. When learning tennis, we allow our motion to be guided by a teacher. In everything we do, we can consciously prioritize certain aspects of our motion to comply with desired restrictions. However, some perturbations happen so fast that we only realize them later on (Kandel et al., 2000; Dhamala et al., 2004). In such cases the body's morphology is able to compensate. For example, when riding a bike on a rough terrain our muscles need to absorb many vibrations, in such a way that our perception and control is not too much affected.

When a robot is able to perform its designated tasks and comply to such externally enforced restrictions, it is called a compliant robot.

1.3 Nuances in compliance

Compliant robots exist in many forms and shapes, some of which are more compliant than others, or are just compliant in a different way. As shown in Figure 1.2 the stiffness or compliance can range from fully rigid to fully compliant. Quantifying this degree of compliance and its contribution to human safety is not straightforward. In the context of the potential dangers associated with industrial robots, the International Organization for Standardization (ISO) introduced a safety standard ISO 10218 in the year 1992 and an updated version (ISO 10218-1) in 2006 to meet new and emerging technologies (Harper and Virk, 2010). Evaluating human safety is not yet standardized when collisions can occur at higher speeds than defined by ISO 10218-1 (Haddadin, 2014). However, in order to evaluate the safety performance of a robot, currently the head injury criterion (HIC) (Bicchi and Tonietti, 2004; Versace, 1971) and abbreviated injury scale (AIS) (Gennarelli and Wodzin, 2006) are most commonly used.

Although the development of compliant robots is mostly driven by the safety properties of physical human robot interaction, it is not the motivation of this dissertation. As illustrated in Figure 1.2, the motivation of this PhD thesis is the controllability of compliant robots for which it is hard or even impossible to determine model information. It does not mean that model-based controllers are rendered useless. On the contrary, as I will discuss later on, they have demonstrated to be very suitable for controlling compliant robots. However, for some robots the degree of compliance might cause difficulties for model-based controllers at which model-free controllers could take over. They are in fact complementary, and depending on the task requirements, robot design, dynamic properties or other requirements, one class of controllers can be chosen over the other or even combined.

For any degree of compliance, between being fully compliant and fully rigid (not included), there exist two types of compliance: *active compliance* and *passive compliance*. In the following I will discuss these types briefly.

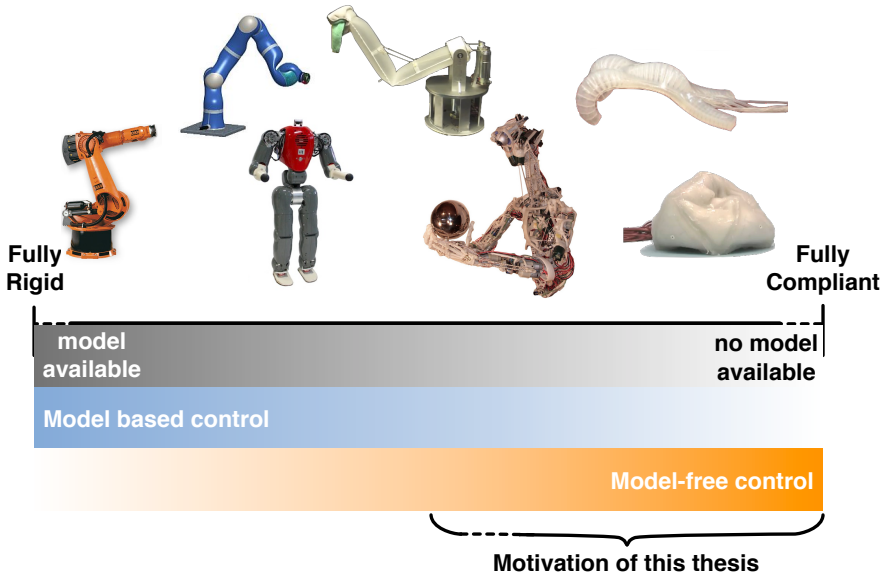


Figure 1.2: Illustration of some example compliant robots located on a scale ranging from fully rigid to fully compliant. The applicability of model based and model-free controllers is depicted on this range and corresponds to the availability of a prior model. The motivation of this thesis is situated at the right part of this illustration, where no or inaccurate model information is available.

1.3.1 Active compliance

Active compliance is attained when the control of a rigid actuator is adjusted in such a way that a seemingly compliant behavior is achieved. In order to illustrate this idea, the example shown in panel **A** of Figure 1.1 is used. The motor is driven by a controller which applies a certain current allowing the measured *encoder*[1] value to correspond to the desired position. By measuring the current on the motor (notice the bidirectional arrow between controller and motor), the controller gets a sense of the amount of torque applied on the attached bar. When the controller adapts its control based on these measurements, this actuator is actively controlled to be compliant.

Several approaches have been proposed to implement and improve active compliance on actuators. Majority of actuators are electro magnetic actuators, which offer low output torque with high speeds. Most applications require some sort of transmission to improve the torque requirements. However, these transmissions cause friction and an increased inertia, making it harder to back-drive[2] the motor. Asada and Youcef-Toumi (1987) developed a direct drive actuator where the transmission between the actuator and the moving part of the robot is removed. As a result, it became possible to use current information as an accurate torque measurement. Due to their increased size to ensure sufficient torque, their applicability is, however, limited. Townsend and Salisbury (1989) uses cable transmissions in order to move most of the actuators out of the moving parts of the robot, providing a low-friction and light-weight solution. The whole arm manipulator (WAM) (Townsend and Salisbury, 1993) or the PHANToM (Massie and Salisbury, 1994) for example, use such a transmission. However, from personal experience with the WAM, slack in these cables (caused by extensive use) makes it much harder to accurately control the robot. Other type of actuators such as pneumatic and hydraulic actuators have their own advantages and disadvantages. For example, the compressibility of air makes a pneumatic actuator intrinsic compliant but as a result less accurate than using hydraulic actuation. Using hydraulics, on the other hand is less safe because of its non-compressibility. Furthermore, both approaches suffer from thermodynamic effects. Morrell and Salisbury (1998); Zinn et al. (2004) proposed to use a combination of pneumatic (high torque) and electromagnetic (low-torque) actuators[3] to compensate for the inaccuracies caused by the compressibility of air. The Hydraulic Quadruped (HyQ) robot by Semini et al. (2011) is a quadruped robot that uses hydraulic actuation to achieve a high power to weight ratio in combination

[1] Measures the rotation angle of the motor shaft.

[2] When moving a compliant robot by hand, external forces are induced on the actuators. Often, the resulting encoder and/or torque changes are measured and used as a teacher signal. This reversed use of an actuator is called back-driving an actuator.

[3] The electro motors can be placed in parallel to the pneumatic actuator or distributed to other parts of the robot.

with electric motors placed in the joints where high velocity control and size is important, but the power to weight ratio is less important. A more detailed discussion of the different actuation approaches is given in Ahmed (2011).

Instead of using torque/current measurements at the actuator side, Vischer and Khatib (1995) proposed to use torque measurements on the moving part of the robot. In combination with a low gear ratio transmission, a high back driveability was achieved.

When *stiffness control* is implemented, a static behavior of the interaction is achieved, meaning that there exists a static stiffness that does not depend on the velocity or acceleration of the motion. A more demanding objective is that of *impedance control*, where a dynamic behavior is achieved (Siciliano and Khatib, 2008). By implementing a second order mechanical system, e.g., a spring damper, within this controller, it becomes possible to absorb introduced perturbations similar to the effect of a spring damper system within a car. A nice overview of force control and its use in compliant robot control is given in De Schutter et al. (1998).

Active compliant actuators improve the safety in human-robot interaction because the robot can sense the interaction forces and react compliantly. Furthermore, they can adapt the compliant behavior during operation, depending on the context of the task. However, the effect of such compliant response depends on the reaction time of the controller, which is often referred to as the *control bandwidth*. When the control loop is too slow, the actual interaction forces will increase dramatically as if there was no compliance present (Lefebvre et al., 2005; Kim et al., 2004). Most controllers today are fast enough to react when slow perturbations are induced. For instance, when back-driving the actuator the external forces are applied gradually and relatively slowly. However, when a sudden perturbation is induced, the control loop is often too slow to react. Even though active compliance has clear limitations, it is still widely used because the desired compliant behavior can be programmed in software. However, the main drawbacks of using active compliance can be solved by adding passive compliant elements to the actuator.

1.3.2 Passive compliance

In order to reduce the reaction delay of an active compliant system, passive elastic elements such as springs are added because their mechanical properties allow them to react instantaneously (Chew et al., 2004; Vanderborght, 2010). In panel **B** of Figure 1.1 an example of a Series Elastic Actuator (SEA) (Pratt and Williamson, 1995) is shown which is a typical example of a passive compliant actuator. Here, the motor shaft drives the rotation of the bar via a spring that compensates for sudden changes in the bar's angular position. This spring has a fixed spring constant that can not be adjusted during operation. Furthermore, note that all physical springs have a damping loss which is equivalent to a damper act-

ing in parallel to an ideal spring. In the example of Figure 1.1 (**B**) an encoder is added at the bar's side of the setup such that the actual rotation of the bar can be measured. Given the spring constant and the difference in measured rotation between both encoders gives an indication of the amount of force applied on the bar. Although passive elements react instantaneously, the range in which they can compensate for external disturbances is small because of the physical limitations e.g., length of the spring. Therefore, the combination of passive elements with active compliant control is often used such that larger but slower changes can be partially compensated by an active compliant control (Schiavi et al., 2009). For example, when current measurements are used to determine the shaft torque at the motor side, the actuator shown in panel **B** of Figure 1.1 is a compliant (active and passive) actuator. Furthermore, the use of passive elements within an actuator has the ability to store energy, that can be reused afterwards to reposition the arm without the need of the motor to deliver too much power (Scarfogliero et al., 2009; Van Der Linde, 1998; Vanderborght et al., 2009b). An often used example is that of a jumping kangaroo where the energy stored within the tendons by landing after a jump can be reused for the propulsion in the next jump. Robots equipped with such actuators are often referred to as a *flexible joint robots*

Figure 1.3: COMAN platform developed by IIT.

However, adding passive elements generally reduces the accuracy of the control. For example, consider the desired target angle that the bar should reach as fast and accurately as possible. Because of the bar's inertia, the spring will cause the bar to oscillate as soon as it reaches the target position. If a factory robot would be equipped with such actuators, the accuracy requirements needed to build for instance a car would not be met. Adaptive compliant designs Albu-Schaffer et al. (2008) are being used more recently in which the amount of stiffness can be adapted while still having passive elements to store energy and to react to fast perturbations.

Let us consider again the actuator shown in panel **B** of Figure 1.1 where the compliance due to the spring should be reduced. By actively rotating the motor shaft into the opposite direction of the external disturbance, the actuation becomes stiffer. When the motor moves in the same direction as the disturbance, the stiffness is reduced (Tsagarakis et al., 2009).

Laurin-Kovitz et al. (1991) developed a programmable passive impedance control approach where a non back-drivable actuator was used in combination with

programmable (adjustable) passive elements. This approach reduces the contact instability caused by the feedback loop time delay while keeping the programmability of the passive elements.

Another approach, called Mechanically Adjustable Compliance and Controllable Equilibrium Position Actuator (MACCEPA) (Van Ham et al., 2007), uses a separate actuator to adjust the equilibrium position of the passive element. The second actuator controls the actual motion of the joint by pulling the spring. The torque generated by the joint depends linearly on the compliance and the angle between the actual position and the equilibrium position. As a result, the equilibrium position and compliance, can be controlled independently. This design was extended by Vanderborght et al. (2009a) such that the torque-angle curve can be modified by choosing an appropriate shape that guides the cable between the spring and the actuator that controls the movement.

Tonietti et al. (2005) developed a Variable Impedance Approach (VIA) where the actuators stiffness, damping and/or gear ratio can be adjusted during the execution of the task. As stated in Ahmed (2011), this approach yields a significant gain in safety performance of the robot for safe physical human-robot interaction. This design comes with the cost of control complexity due to the more complex non-linear springs inside the actuation mechanism. The Actuator with Mechanical Adjustable Series Compliance (AMASC), is inspired by the antagonistic actuation used by humans (Hurst et al., 2010). Similar to MACCEPA the compliance and equilibrium position can be controlled independently. However, as with the VIA design, it comes at the cost of control complexity. For a more detailed overview of VIA actuator designs I would like to refer to Vanderborght et al. (2013). The DLR LWR-III arm is constructed with harmonic drives, which are actuators that follow the Variable Impedance Approach. By having such elastic joints that can be adjusted during operation, the robot is able to react fast enough to avoid high impact forces during a collision. De Luca et al. (2006) proposed a collision detection approach that only uses proprioceptive robot sensors. Furthermore, the approach additionally provides directional information for a safe robot reaction after a collision.

Often pneumatic muscles are used because of their inherent passive compliance caused by the compressibility of air. The McKibben muscle (Tondu and Lopez, 2000) is the most well known pneumatic artificial muscle but suffers from hysteresis effects caused by friction and a high minimum pressure to generate any contracting force. The Pleated Pneumatic Artificial Muscle (PPAM) (Daerden and Lefeber, 2001) improves the MCKibben muscle, significantly reducing hysteresis effect and omitting the need of a minimum pressure. Its design does not need a heavy/complex gear transmission allowing for a direct coupling with the joint. For instance, the biped robot called Lucy, developed at the University of Brussels, uses several PPAMs in an antagonistic configuration demonstrating an effective power

Figure 1.4: **A** Depicts the Oncilla quadruped robot which our lab developed together with the Biorob Lab at EPFL. **B** The i-HY robot hand with its flexible fingers. **C** iRobot's Chembot consists of a silicone sphere.

to weight ratio (Vanderborght, 2010). Van Damme et al. (2010) concluded based on his experiments that his pneumatic manipulator was unsafe under PID control. Additionally, he noted that the intrinsic passiveness of a pneumatic actuator is insufficient for safety when potential control errors are present. In fact, the passiveness of the the pneumatic actuator improves safety in some cases of sudden impact, but its ability to store energy can also make it more dangerous. Van Damme et al. (2010) also observed that at the moment of impact, the contact forces are almost independent of the joint stiffness and mostly influenced by the inertia of the robot link.

Within the European AMARSi project, a compact SEA module, shown in panel **C** of Figure 1.1, was designed and developed by the Advanced Robotic department of the Italian Institute of Technology (IIT) (Tsagarakis et al., 2009). This module was later on used within IIT's Compliant Humanoid Platform (Tsagarakis et al., 2013) shown in Figure 1.3, which belongs to the state-of-the-art in compliant humanoid design.

There exist many adaptive compliant actuator designs, of which a detailed overview is given in Ham et al. (2009); Albu-Schaffer et al. (2008).

The trend of compliant robotics is being extended even further to the other parts of the robot. Flexible materials are being used as structural parts of the robot making them more robust against unanticipated disturbances. This type of robots is often called *flexible link robots*. In collaboration with our lab, the Biorob Lab at Ecole Polytechnique Fédérale de Lausanne (EPFL) designed a compliant quadruped robot, called Oncilla (Sproewitz et al., 2011), shown in panel **A** of Figure 1.4. This cat-like robot has passive elements within its legs. A key advantage over its counterparts such as DARPA's little dog (Murphy et al., 2011), is that these passive elements allow the robot to move over irregular terrain without the need of adjusting its control. This also the case for the COMAN robot (Tsagarakis et al., 2013) , where the springs forgive small differences, making the robot more stable. One can argue that the complexity of the required control is split up between the

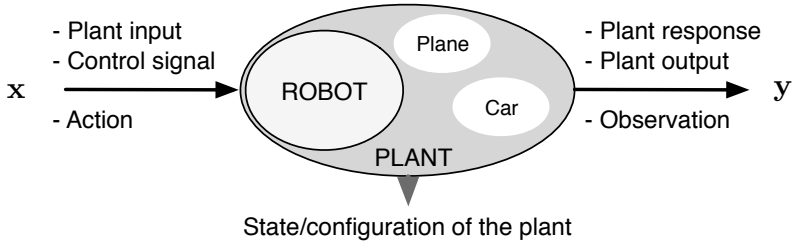


Figure 1.5: An overview of the used taxonomy to describe a control problem.

controller, the body and its environment(Pfeifer and Bongard, 2007). In the Oncilla it is, however, not possible to adjust the stiffness of its springs, making it hard to achieve accurate control when needed. Another example is that of the i-HY hand developed at the Harvard Biorobotics Lab within DARPA's Autonomous Robotic Manipulation - Hardware Track (ARM-H) program (panel **B** of Figure 1.4). It consists of flexible fingers which are able to manipulate a wide variety of objects while reducing interaction forces with its environment (Odhner et al., 2013). iRobot's Chembot (panel **C** of Figure 1.4) is entirely built from flexible materials and selectively changes the rigidity of individual sectors of its silicone sphere such that air within the sphere can deform the robot as desired. Adding all these structural passive compliant elements comes with a cost. The more passive elements, the harder it becomes to control.

1.4 Adaptive control

As mentioned in the previous section, the compliance/stiffness is often adapted during operation (Bicchi et al., 2005). However, reducing the compliance when accuracy is needed is not a good solution because it reduces the safety purpose of having compliance in the first place. A need exists for adaptive control algorithms which allow compliant robots to accurately interact with their environment even though they consist partially or fully out of flexible materials such as silicone or springs of which the dynamic and kinematic properties are hard to determine.

Before continuing, I will briefly give an overview of the taxonomy used concerning robot control or plant control in general. As shown in Figure 1.5, The input which drives the plant is denoted by **x** while the response to this input is denoted by **y**. The state of the plant describes the plant in terms of parameters at a particular moment in time. For instance, the motion of a train on a rail is fully defined by its position and velocity. In some control tasks this state is (partially or fully) observable and is part of the observed plant output.

An adaptive controller tries to estimate uncertain plant and control parameters during operation by using measured plant responses (Åström and Wittenmark, 2008).

Research in adaptive control started in the 1950's, mainly to design autopilots for military aircrafts which are inherently unstable. However, this interest diminished after the crash of an X-15 experimental aircraft equipped with an adaptive controller which produced too high gains after going into a pilot induced oscillation. As a result the pitch angle of the aircraft started to oscillate with an increasing amplitude, eventually causing the airplane to crash and killing the pilot. During the last decade, however, many new approaches emerged with many practical applications as result. For example, the use of gyroscopic stabilizers and inertial measurement units has enabled the development of drones.

In the following sections I will discuss the benefits of having an adaptive controller and how it can be applied with or without prior information about the robot that needs to be controlled. For a more detailed overview of this separation between controllers that use prior information, and others that do not, I would like to refer to Rigatos (2009).

1.4.1 Model based control

Model based control methods forms a class of control strategies in which prior knowledge about the plant (e.g., robot) is used. This prior knowledge can be represented by an equation which describes the evolution of the plant's state and/or knowledge about physical properties such as mass, geometric information, mass distribution, among others. The collection of all this information is called a model of the plant. Having a model does not mean that it accurately describes how the plant evolves over time. However, one of the advantages of having a model-based controller is the ability to reason about the underlying physical properties to for example identify possible causes of control problems more easily.

The control of linear systems has been extensively studied (Kwakernaak and Sivan, 1972; Lewis et al., 2012). However, more complex and nonlinear systems are hard to model correctly. Due to friction, heat or other phenomena, which are not taken into account, the model is an approximation of the real physical system. Instead of having complete model information in the form of analytical equations, it is possible to create an model approximation prior to control. This is a data driven approach called system identification where the plant is identified by acquiring and processing raw data. One can then choose from a set of function approximation algorithms that learn the system model. These models are of course approximations of the real physical plant. In order to improve the accuracy, feedback information is often incorporated to compensate for the modeling errors. This can be written

more formally as:

$$\text{control input} = \text{model based control} + \text{feedback control} \tag{1.1}$$

Often a Proportional-Integral-Derivative (PID) controller is used to reduce the control error. A PID controller only uses a measurement of the control error to generate an additive control signal which is sufficient to bring the measured error down to zero. This controller is model-independent and requires only a limited set of parameters to be tuned. This is also one of the main reasons why a PID controller is widely used in industrial applications. For compliant control, however, it is less useful because during a sudden perturbation of the robot's motion the control error would increase fast, causing the PID-controller to generate a proportional gain and more importantly a derivate gain with large interaction forces as a consequence. In order to prevent such large interaction force, a more intelligent feedback controller is necessary. An example is Active Disturbance Rejection Control (ADRC) (Han, 2009), which is a robust control approach that extends the model of the plant with an extra state variable that represents everything for which the internal plant model is insufficient, similar to how the above PID controller compensates for inaccurate model information. Albu-Schaffer et al. (2007) proposed a control framework for flexible joint robots which incorporates gravity compensation, and a desired Cartesian stiffness relation based on the joint angles. In Van Damme et al. (2009) a proxy-based sliding mode controller was presented to reduce the slow response to position errors when using a pneumatic manipulator. In Todorov et al. (2010) a parametric model identification was performed by fitting the model to experimental data and optimizing model-based feedback controllers. In Tsagarakis et al. (2009) a PD controller is used in combination with a compliance regulation filter that allows the compliant actuator to be controlled with a desired impedance setting. This control compliant regulation filter takes into account model information of the used springs and the inertia and axial damping coefficients of the link. These are just a few examples of model-based controllers that are able to control robots with flexible joints. Many more controllers exist some of which achieving remarkable results (Agachi et al., 2007; Brosilow and Joseph, 2002).

1.4.2 Biologically inspired control

Humans and biological systems in general are able to develop advanced motor skills, allowing them to interact compliantly with each other, constantly anticipating an uncertain environment. Biological research has partially unveiled the underlying processes that makes such interesting control possible. Inspired by these biological findings, scientists and engineers are building robots, often built from compliant materials, which are driven by biologically inspired control approaches, e.g., the salamander-like robot, called Salamandra Robotica 2 and shown

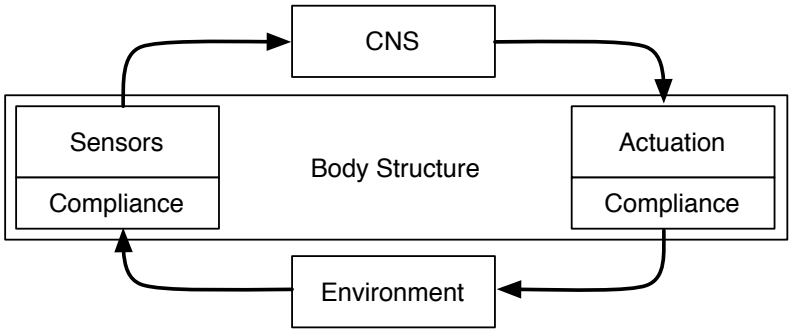


Figure 1.6: An abstraction of how the Central Nervous System (CNS), the body and the environment are interacting with each other.

in Figure 1.7,**A**, developed by researchers of the Biorob Laboratory (EPFL) under guidance of Auke Ijspeert (Crespi et al., 2013). The control of this robot is based on coupled oscillators to mimic Central Pattern Generators (CPGs) found in the spinal cord of biological systems. Furthermore, this controller is distributed over the entire robot in such a way that separate segments can move independently. The same research group designed the Cheetah Cub (shown in Figure 1.7, panel **B**), also driven by a coupled oscillator that enables the robot to reach speeds up to 1.42 m/s (Sproewitz et al., 2013). Its brother, the Oncilla quadruped robot (Sproewitz et al., 2011) which I discussed before, has more degrees of freedom, allowing it to move all legs sideways as if the robot had shoulder blades. In the last 10 years, the development of biologically inspired robots and control has gained a lot of interest within the robotics community, yielding many intriguing robots of which the Eccerbot (Potkonjak et al., 2010) (tendon driven humanoid) and the octopus robot (Margheri et al., 2012) are a few examples.

In Figure 1.6, I have tried to make an abstraction of how one interacts with its environment. In general, one can state that based on sensory information the Central Nervous System (CNS) actuates the body, resulting in an environmental change that in turn affects the information flow. From a philosophical point of view, the compliance is acting here as an interface between the body and the environment. At the actuation side it allows for the body to act upon the environment robustly, forgiving small errors in the CNS's anticipation of the environment. At the sensory side, the compliance is filtering/preprocessing the information flow from the environment to the CNS.

The CNS is a flexible entity that tries constantly to distribute and outsource the complexity of a task over multiple elements. Even though it constantly tries to simplify its use, the most adaptable element is the CNS itself. The environment can also be changed relatively quickly. For example we use tools (e.g., a

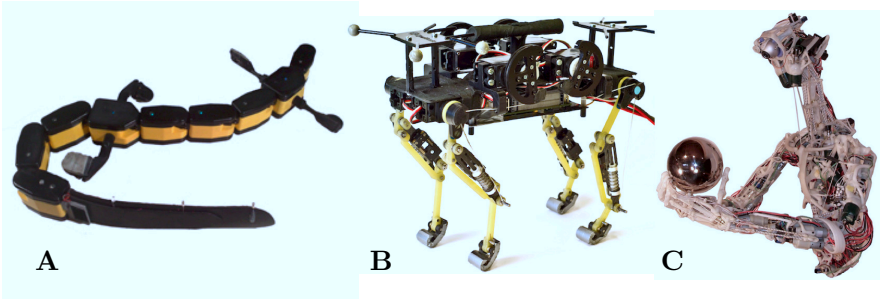


Figure 1.7: An overview of robots with biologically inspired controllers and/or mechanics. **A** and **B** are the Salamandra Robotica 2 and Cheeah Cub, respectively, developed at the Biorobotics Laboratory of EPFL. **C** Tendon driver robot called Eccerbot.

pen, a knife, roads, cars, computers) to simplify tasks. However, some environmental phenomena (possibly caused by our collective) are, due to their complexity and our lack of understanding, not yet controllable. When these environmental changes are insufficiently handled by our compliance and affect our survival, the body will start to adapt through the process of evolution. However, such adaptation takes place over the course of several generations and is therefore much slower than the duration at which the CNS can adapt itself or parts of the environment. The changes of the body are thus directly influenced by the environment and indirectly by the CNS. Our CNS is in turn affected by our body and environment. It is this interplay between the CNS, body and the environment which is investigated in the domain of *morphological computing* (Pfeifer and Iida, 2005). However, based on the above reasoning this interplay is working at different time scales, allowing one part to adapt without being to much affected by the other. Therefore, in my opinion, simplifying the control complexity by modifying a compliant robot's morphology should be investigated from an evolutionary point of view and thus a slower time scale. In contrast, when a certain morphology is given, the control should be learned by interacting with the robot and its environment.

Developing a biologically inspired control hierarchy for such compliant robots with highly non-linear behavior that is able to adapt to dynamical changes arguments the need of a model-free approach that can partially[4] or fully learn the unknown properties.

[4]In combination with a model based approach.

1.4.3 Model free control

In model free control, the assumption is made that no prior knowledge about the plant is available, e.g., when the dynamics are too complex or when the physical process is not well understood. Often the analogy with a *black box* is used because the internal processes of the plant are unknown. One can only observe the control signal going into the black box and the response coming out. In order to make control possible, information about the plant is gathered by interacting with the plant itself. Due to its model independence, a model free controller is applicable to a wide variety of tasks without the need for complicated manual tuning of parameters or modifying them when the plant dynamics change during operation. A model-free adaptive controller is in fact a nonlinear dynamical system with online (during operation) parameter estimation. If the controller is calculating control parameters based on estimations of the plant parameters, this controller is called an *indirect adaptive controller*. The performance of such a controller depends on the convergence of the estimated plant parameters to their true values. When only control parameters are estimated, such a controller is called a *direct adaptive controller*. Their exists a wide variety of model free control approaches, for instance, the above mentioned PID controller or the Model Free Adaptive (MFA) control by Cheng (2000) for plants with slowly changing states or a modified versions of MFA (Karoń, 2012) for rapidly changing states. This MFA controller uses a multilayer perceptron neural network that is trained online, which makes the controller inherently robust for changes in plant behavior or other uncertainties. Yazdizadeh et al. (2000) demonstrated that his proposed neuro-dynamic structure was able to identify highly nonlinear without any a priori information about the nonlinearities of the system and without any off-line training. In Su and Khorasani (2001) a control approach is presented where a classical rigid body inverse dynamic model approach is used, but where the unknown non-linearities are learned by neural networks. This is a nice example of how model-based and model-free approaches can be complementary used to get the benefits of both sides. In Section 1.5.5 I will explain what a neural network is and describe its use for control. In Jalali et al. (2013) a sliding mode controller is optimized using particle swarm optimization and used to control a robotic manipulator.

The main assumption made within this dissertation is that no prior knowledge about the robot or plant is available. This knowledge concerns the physical properties (e.g., mass, density distribution, geometric properties) of the plant. However, when other information is used this will be mentioned explicitly.

In order to make model-free adaptive control possible, I have employed techniques from the domain of machine learning. In the following section, an introduction to machine learning is presented in which I highlight some important concepts on which the proposed ideas in the remainder of this dissertation are built.

1.5 Machine learning

There exist many definitions that try to describe what Machine Learning (ML) actually is. For instance, Samuel (2000), one of the pioneers in the field of computer gaming and self learning programs, defined ML as a *"Field of study that gives computers the ability to learn without being explicitly programmed"*. He created a checkers program that could learn from experience and so improved its performance by playing against itself, eventually surpassing Samuel's own chess abilities. A more modern definition by Mitchell (2006) describes ML as follows: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"*. Often ML and Artificial Intelligence are mistaken for being one and the same thing. However, AI as we know it from science fiction stories such as the HAL 9000 in Arthur Clarke's book Space Odyssey, is defined as an intelligent, self thinking, machine, often assigned to an entity/agent. Its rather hard to define when such an entity exhibits autonomous thinking or intelligent behavior. ML on the other hand uses a more scientific taxonomy in which the performance of a learning program can be measured objectively instead of trying to relate to human-like behavior.

1.5.1 Creating a model approximation

Within Mitchell's definition of ML, experience is represented by observations that are the result of an underlying unknown process. Consider for instance noisy velocity and altitude measurements made during the final minutes of the approach landing of NASA's Curiosity, shown in Figure 1.8 and indicated by black circles. Now suppose that we want to know the velocity of Curiosity for every possible altitude and not only for the ones that were measured. The measured data provides us with observations from which one can learn the underlying relationship between the measured velocity and altitude. Such a relationship is represented by a model that tries to describe the data as well as possible and allows us to generalize to unseen information. Actually this is very similar to what humans do on a daily basis. We try to find patterns in everything we perceive. For instance, when reading a book we are provided with descriptions of characters and details about their surrounding from which our imagination generates a model that fills in the missing pieces.

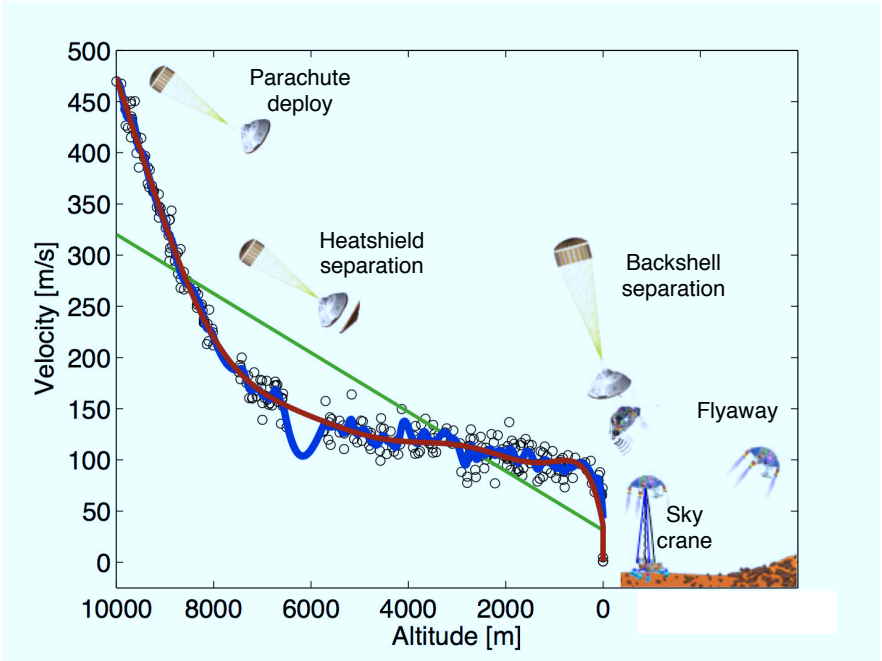


Figure 1.8: This illustration shows altitude and velocity measurements based on little data published on the landing of NASA's Curiosity. This published data was augmented with fictive data to suit the purpose of having a good regression example. The data point indicated by black circles represent the velocity and altitude measurements. The green and blue line/curve indicate a model which under-fits and over-fits the data, respectively. The red curve represents a model that generalizes velocity approximations well to unseen altitudes. The actual phases of the landing are shown as well, explaining the non-linear relationship between both altitude and velocity measurements.

1.5.2 Representation vs generalization

When a model is learned based on measurements, the generalization capabilities for unseen information depend on the complexity of the system. As shown in Figure 1.8, fitting a line to the observed measurements (shown by a green line) does not give good results. Such linear regression on the data is insufficient to explain the data itself and to generalize to unseen altitudes. This problem, where the complexity of the model is too low compared to the number of data points, is called *under-fitting*. Having a model that is too complex, in contrast, is also not a good solution because the model will describe the noisy measurements too well, yielding bad generalizations for unknown altitudes. This problem of *over-fitting* the measurements is depicted by the blue curve in Figure 1.8. So there exists a trade-off between representing the observations and having a model that generalizes well.

The red curve shows a possible good fit. The performance of such a fit needs to be determined by comparing the model approximation to an actual measurement. When new measurements, not used to fit the model, are unavailable the original dataset should be divided into two parts: one part to fit the model and another to test the model. In the remainder of this dissertation, fitting a model to a set of data is called *training a model with data*. When the performance of the model depends on a certain parameter that should be optimized for the task at hand, the data is divided into 3 parts, one to train the model, one to validate the parameter and one to test the model with the optimized parameter. In Section 2.1.5.3 this process is explained in much more detail.

1.5.3 Learning approaches

In the example described above we had both altitude and velocity measurements available from which we could train a model to infer the velocity given the altitude, or the other way around. However, in some learning tasks, there are no training examples available that contain the desired model output. Based on this distinction, the learning approaches in ML can be classified as follows:

- **Supervised learning:** The training data in this case consists of example input and desired output data such as for the Curiosity example above. A supervised learning algorithm will produce a approximation of the output corresponding to a certain input, based on the training data. Most algorithms described in this dissertation are of the supervised learning type.
- **Unsupervised learning:** As the name suggests, the data now only contains observations of the input. There is no data available about the desired output of the algorithm. Such algorithms try to find hidden structure within the available data without examples on how to achieve this. Clustering algorithms (e.g., k-means) are an example of unsupervised learning algorithms. Here, the centers

of a number of clusters are chosen randomly. Each data sample is assigned to a cluster, depending on its distances to those centers. Whenever a new data sample is added, the closest cluster center is updated in such a way that it moves more towards the newly added data point.

- **Semi-supervised learning:** Having a dataset with pairs of example input/target pairs is often hard to realize. Within the class of semi-supervised learning algorithms the training data is similar to the unsupervised approach. However, a few examples of the desired output are added as well. The idea here is that a model can be trained on the few data samples for which an example output is available, and that this model can be improved by the data for which no desired target is accessible.

- **Reinforcement learning (RL):** Is similar to unsupervised learning. Here, no detailed description of the desired output is available, but a general quality measure can be given. The learning algorithm now receives a reward signal that evaluates the performance of the generated solution. Based on this reward, the algorithm can be adapted. RL approaches are often used in robotic applications where a number of actions needs to be taken to solve a certain task. As it is unknown how to solve this task, the performance of the model can only be evaluated at the end (after all necessary actions have been taken). In Section 6.6 I will explain RL into more detail.

1.5.4 From linear to non-linear regression

As shown before, fitting a line to measurements of Curiosity's landing is insufficient to create a mapping that approximates the velocity v for a certain altitude a because there exists a non-linear relationship between them. Finding the underlying non-linear relation seems hard. However, by non-linearly transforming the altitude measurements, one can still use a linear regression method to find a model. More formally we want to find a function $\Phi(a)$ that transforms the altitude measurements a in such a way that the velocities v are linearly related to $\Phi(a)$. A simple example of such a function would be $\Phi(a) = [a, a^2, a^3]$ or $\Phi(a) = [a, \sin(a), a^2]$. Instead of having one value to correlate v with, we have 3 values in both examples. Whether or not this increase in dimension is sufficient to find a good linear model, depends on the chosen features of a (square, sine, ...). Which type of *features*[5] (e.g., polynomials, Gaussians, ...) are good in turn depends on the data and is quite specific for the algorithm that is used. Well known non-linear regression approaches such as Gaussian Process Regression (GPR), Gaussian Mixture Regression (GMR), Support Vector Regression (SVR) and many others can be formulated as being a linear

[5] Is an individual measurable property of the process that is being observed.

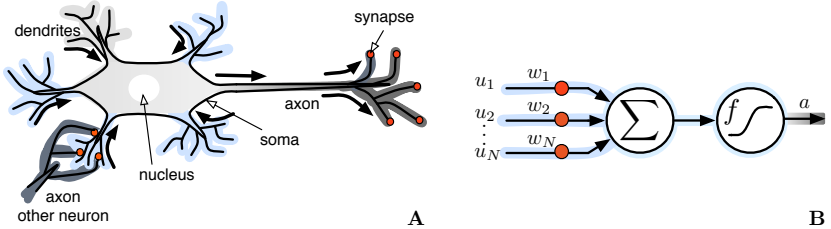


Figure 1.9: This illustration depicts a biological neuron **A** and its corresponding mathematical representation **B**. The synapses (red circles) are represented by weights which modulate the input signals of the neuron.

model on a set of non-linear features which are calculated based on the input data (altitude measurements in the used example).

Another well known approach is called Artificial Neural Networks (ANN) and uses a set of processing units that are connected with each other, similar to how neurons are interconnected in the brain. Like other machine learning approaches, ANNs have been successfully applied to solve a wide variety of tasks, including fault detection and diagnosis in industrial processes (Maki and Loparo, 1997), tornado prediction (Marzban and Stumpf, 1996), optical character recognition (Avi-Itzhak et al., 1995) and non-linear flight control using neural networks (Kim and Calise, 1997) among many others.

In this dissertation I use a modeling approach called Reservoir Computing (RC), that is based on concepts of Artificial Neural Networks. I will therefore introduce ANN in much more detail before giving a thorough description of RC in Chapter 2.

1.5.5 Artificial neural networks

In order to interact with an environment, sensory information is processed and a possible response is transfered to the corresponding actuators. Humans and animals use a central nervous system for the information flow and processing. The brain is the most complex, non-linear and not yet fully understood part of this central nervous system. It is located close to the primary sensory organs to have fast processing of visual and auditory information, among other reasons. The brain consists of glial cells, used for several functions such as structural purposes, and neuron cells, called neurons. The human brain contains in total 15-33 billion neurons. As shown in panel **A** of Figure 1.9, a neuron gathers signals from other neurons through tree like structures called dendrites. The soma of the neuron processes these signals and transmits a response along its axons. These axons can carry the signals over long

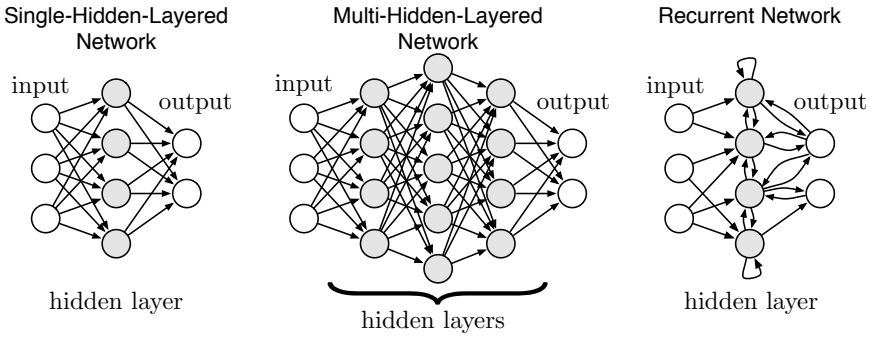


Figure 1.10: This illustration depicts examples of a single-hidden-layered, multi-hidden-layered and recurrent network architecture. The white circles are input or output nodes while the hidden nodes are represented by gray circles.

distances, targeting distant cells of the brain or body. The axons are connected to dendrites of other neurons through synapses. Every synapse can modulate, or even obstruct, the signals passing through. A large number of these synapses modulate the signals depending on properties of the signal itself, giving the brain the ability to reuse structures for different purposes. It is believed that such modifications are associated with memory and learning (Shepherd et al., 2004). This insight into the brain's learning mechanism inspired McCulloch and Pitts (1943) to create a simplified mathematical model of a brain. They proposed an artificial neuron of which an illustration is shown in panel **B** of Figure 1.9. The corresponding equation is formulated as follows:

$$a = f\left(\sum_{i=1}^{N} w_i u_i\right), \tag{1.2}$$

where u_i represent the input signal from a particular dendrite which is modulated by a synapse, represented by a weight w_i. The result of processing all the input signals in the soma is simplified by a non-linear *activation function* f. The axon output is denoted by a. Originally McCulloch and Pitts (1943) proposed to use a threshold function for f, which is high when the accumulation of the signals reaches a threshold, or low when it doesn't. However, other activation functions have been shown to be useful as well.

Within the artificial intelligence community, there existed a large interest in such artificial neurons because, similarly to a brain, such neurons can be interconnected to form a network-like structure, an Artificial Neural Network (ANN), that is capable of solving a large number of tasks. The simplest network structure is called a single hidden layer feed-forward architecture, shown in Figure 1.10. In this feed-forward neural network the information propagates in one direction from the input

through the single hidden layer to the output. The layer in the middle is called a hidden layer because only the inputs and outputs of the network are observed. Instead of having a single hidden layer it is possible to have multiple hidden layers where one layer is connected to the next one (Figure 1.10). When each layer (input, output and one or more hidden layers) is fully connected to the next one and each neuron's activation function is non-linear, such a network is called a Multi-Layer Perceptron (MLP) (Rosenblatt, 1961) and it is proven to be able to approximate any continuous function of real variables arbitrarily well depending on the number of neurons (Cybenko, 1989). This means that when too many neurons are used, it becomes possible for the MLP to over-fit the training data. Training such networks requires that every weight w_i that corresponds to a synapse is adapted, such that the desired function is approximated. One of the many training algorithms is called Back-Propagation (Rumelhart et al., 2002). It essentially propagates the error between the approximation and the desired output backwards through each layer, adapting the weights appropriately. One of the main criticisms about this approach is its slow convergence and the fact that it is not guaranteed to find a globally optimal solution. After the publication of a book by Minsky and Papert (1969), the advances in neural network research stagnated because it revealed that a single hidden layer network is incapable of learning an exclusive-or function and that the computers at the time were not advanced enough to handle long running times needed to train large neural networks. As the development of computing systems advanced, the interest in using NN increased again. Also within the control-related research community the use of NN became more and more popular (Cessac, 2010). For instance, Kawato et al. (1987) used a NN to change the motor commands by predicting the possible errors of a movement. Since the publication of Narendra and Parthasarathy (1989), the use of NNs for identification and control of nonlinear systems has gained a lot of interest. For instance, Nguyen and Widrow (1989) used a NN to control the truck backer-upper[6] problem. In Hung and Chung (2007) and Spooner and Passino (1996), NNs are used in combination with a classical Sliding Mode Control approach. Other approaches such as Ge et al. (2008) and Yang et al. (2008) use a NN to train a predictor that is used to construct output feedback control. In these works, NNs are used as static function approximators or, when their input is a tapped delay line[7], to model a finite memory functional dependence.

With the advent of deep learning approaches, the use of NN became popular again. Deep learning is a trendy word for an architecture were multiple hidden layers are stacked upon each other. Hinton et al. (2006) showed that each hidden layer could be trained separately by treating each layer as a restricted Boltzmann machine, after which the training is fine tuned with back-propagation. Although

[6]Here a truck needs to be driven backwards given a desired target position.

[7]At any moment in time the network input contains several time instances of the input signal. For instance, the network input contains the current and previous value of the input signal.

this publication generated much interest, the underlying concepts themselves are not new. For instance, the Neocognitron architecture by Fukushima (1980) introduced a stacked neural network to build a pattern recognition mechanism. Furthermore, Schmidhuber (1992) already demonstrated that it is possible to train a stacked network layer by layer in combination with a final optimization step with back-propagation. In this publication, however, each layer is a Recurrent Neural Network instead of a feed-forward NN.

1.5.5.1 Neuron and network state

Previously I defined a system/plant state as being a set of parameters that completely describe the system at a particular moment in time. Given this general description, a neuron state can be defined as follows: A neuron is completely described by its activation function and when this activation function depends on time, the value of such activation function defines the *neuron state*. The network, on the other hand, is completely described by all neuron states and the connections between them. When these network connections are time independent the current *network state* is defined by the current state of all the neurons.

1.5.5.2 Recurrent Neural Networks

Feed-forward NNs are useful to create a static mapping between the network output and input. The mapping of this network does not depend on time, preventing the network to sustain a certain degree of contextual information. However, many modeling tasks, such as time series prediction or unsegmented connected handwriting recognition (Graves et al., 2009), exhibit a dynamic temporal behavior. A workaround for feed-forward NNs is to use a time window such that information from previous time steps is incorporated into the static input-output mapping. A more natural and biological plausible way is to add internal feedback connections, called recurrent connections, into the feed-forward network causing the neuron states of the network to depend on previous states. In other words, such *Recurrent Neural Networks* (RNNs) exhibit a memory-like behavior due to these recurrent connections.

Within the field of non-linear control, the RNNs also provide a more natural and richer alternative to using tapped delay lines or NNs. These RNNs have been used very successfully to control non-linear dynamic systems (Levin and Narendra, 1996; Su and McAvoy, 1997; Pan and Wang, 2011). In recent work, Prokhorov (2007) superbly demonstrates the very rich modeling capabilities of RNNs in a neurocontroller for the electric throttle of a hybrid vehicle. Although several neurocontrollers incorporate some prior knowledge, some approaches achieve control without any prior information about the plant or robot. For instance in (Wang and Hill, 2006) and (Wang and Hill, 2007), an adaptive neural controller is used to em-

bed the unknown system dynamics into the control process. In (Chow and Fang, 1998), two RNNs are used: one for approximately modeling the nonlinear plant, the other to control towards the desired plant response.

Training such RNNs is a bit more complicated than training a feed-forward neural network. However, Werbos (1990) developed a generalization of the Back-Propagation algorithm of Rumelhart et al. (2002) based on the following insight. An RNN can be represented by a single hidden layer where the input also includes the previous neuron states. These previous states can in turn be represented by again a single hidden layer that depends on the previous input and neuron states. An RRN can thus be unfolded over time such that it resembles a multi-layered feed-forward NN. The number of hidden layers within such an unfolded RNN depends on the number of time steps of the training data. The error on the output can again be propagated backwards through the unfolded network and thus time, hence the name for the entire process: Back-Propagation-Trough-Time (BPTT). However, this algorithm is based on the stochastic gradient descent learning approach, causing BPTT to suffer from the same weaknesses: slow convergence times and the possibility to get stuck into a local optimum (not the overall best solution). Additionally, there exists the possibility that gradient information calculated with BPTT fades away due to the number of hidden layers which introduce a large number of non-linearities between the past and present. This fading gradients problem, and the regular bifurcations encountered during training using stochastic gradient descent (Bengio et al., 1994; Pearlmutter, 1995; Suykens and Osipov, 2008) make RNNs notoriously difficult to train.

1.5.5.3 Reservoir Computing

To overcome the difficulties of training a RNN, researchers have found an alternative solution. Instead of training all the connection weights of a RNN, only the connections to the output layer, often called *readout*, are trained. All other connection weights, are randomly initialized and kept fixed during training. This approach simplifies the training of an RNN such that any regression or classification approach can be used to train the readout layer. Often, linear approaches are used because they ensure convergence and yield a globally optimal solution. When a linear method is applied, the neurons' states within the RNN provide features which are non-linear transformations of the input data. This is similar to the underlying mechanisms of most non-linear regression methods, as described in Section 1.5.4. This concept of simplifying the training of a RNN has been introduced independently by Jaeger (2001) as Echo State Networks (ESNs), Maass et al. (2002) as Liquid State Machines (LSMs) and a few years later by Steil (2004) as Back-Propagation Decorrelation, all under different names due to their subtle differences. For instance, ESNs are used for analog neurons, while LSMs are more general and can

be used for both analog and spiking neurons[8]. Due to their similarity, Verstraeten et al. (2007) unified these three approaches under the name Reservoir Computing (RC), which is now commonly used. The randomly connected network within a RC-system is called the *reservoir*. For a detailed overview I would like to refer to Lukosevicius and Jaeger (2009). Within this dissertation I will only consider the ESN approach, of which a detailed description is given in Chapter 2. The insight that an ESN maps the input onto a high-dimensional feature space, more specifically onto the reservoir states, allows us to consider the reservoir as a spatiotemporal kernel of which the feature space is calculated explicitly. Hermans and Schrauwen (2011) extended this idea to infinitely sized recurrent neural networks by using recursive kernels. ESNs have been proven to be successful on tasks such as time-series prediction (Jaeger and Haas, 2004; wyffels and Schrauwen, 2010), speech recognition (Skowronski and Harris, 2004; Triefenbach et al., 2010), epileptic seizure detection (Buteneers, 2012) and robot navigation (Antonelo, 2011). In Verstraeten et al. (2005) stochastic bitstream neurons were used to build a reservoir system suitable for hardware implementations. In fact, a reservoir does not need to be a neural network. Any dynamical system creates a feature mapping from a given input. For many of them, the state space dynamics can be tuned in such a way that this mapping becomes useful for computations. This allows the concept of Reservoir Computing to be applied on physical systems such as analog electronics (Schürmann et al., 2004; Schrauwen et al., 2008), optical computers (Vandoorne et al., 2008), biological neural tissue (Nikolić et al., 2006), or a water surface (Fernando and Sojakka, 2003). Considering Pfeifer and Iida (2005), robots can also provide a similar mapping to a high dimensional state space, allowing the morphology of the robot to be used for computational purposes. Caluwaerts and Schrauwen (2011) showed that high-level environmental information can be extracted form the state of a tensegrity robot. Nakajima et al. (2013) demonstrated that the body of an octopus robot is behaving as a reservoir with which computations can be performed.

1.5.6 Controlling a robot

In order to learn a model, every modeling approach such as an ESN needs to be provided with data (e.g., observations and experience). However, a robot or plant in general, is a real world system on which the physical laws act in continuous time. The controllers that drive these systems, on the other hand, are implemented on digital computing units which operate in discrete time. This difference in time domain should be taken into account when controlling a plant. As shown in Figure 1.11, the observations of the plant response to a control signal are made by a sensor. This

[8] A spiking neuron represent a more accurate neuron model where the neuron activity depends on the number of spikes that enter the neuron.

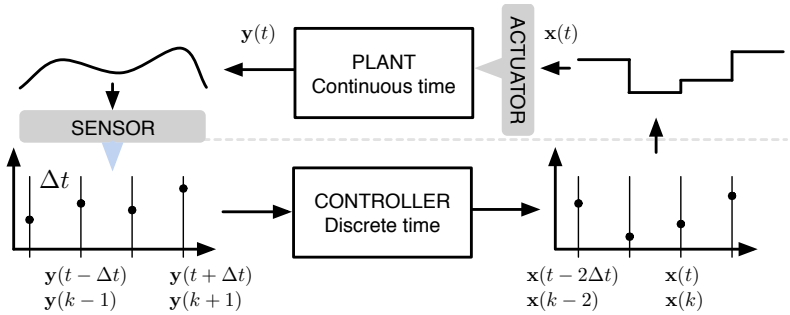


Figure 1.11: Schematic overview of how continuous time information from the plant is observed by the controller, and how the discrete time control signal is converted to a continuous time signal which drives the actuation of the plant.

sensor converts the continuous time observation to a discrete version by sampling the observations every Δt. In this dissertation, Δt represents the control period at which the plant is controlled. This corresponds to the rate at which sensor samples are used. To indicate the difference in time domain, t is used for continuous time and k for discrete time. They are related to each other through:

$$t = k\Delta t \quad \text{such that} \quad t + \Delta t \text{ corresponds to } k+1 \tag{1.3}$$

After recording observing the sensor value, the controller calculates a control command which is converted to a continuous time actuator signal. This actuator signal is constructed by keeping the calculated discrete values constant during an entire time step Δt (zero-order hold). This means that a new control command is sent every time step Δt. When controlling a real robot, the time step size is important because the controller can not take longer than Δt to produce a new command (real time constraint). Increasing the control period is not a good solution if fast responsive behavior is required. When using a simulation model, however, it is possible to halt the integration of the physical simulation environment during which the controller can take as much time as needed to calculate a new control signal. As a result, it is possible that complex control algorithms that work well in simulation do not necessarily work well in real time. Therefore, control approaches that can benefit from parallel computations, e.g., Echo State Networks, are better suited.

1.6 Dissertation structure and contributions

Reservoir Computing (Chapter 2)

A Reservoir Computing approach (RC) is a technique that efficiently trains a Recurrent Neural Network by only training the output connection while keeping all other connections fixed. Although there exist several RC approaches, only the Echo State Network (ESN) technique is considered. As most approaches within this dissertation depend on ESNs, this chapter provides a detailed description of the technique, the influence of its most important parameters and different training algorithms that can be applied.

Motion Pattern Generator (Chapter 3)

This chapter gives a brief introduction to dynamical systems and how the dynamics can be exploited to embed motion demonstrations. An ESN based Motion Pattern Generator (MPG) is described that can generalize beyond the motion demonstrations. Although the embedding of periodic motions existed before, I extended its use to learn multiple discrete motions into a single ESN. Additionally, it is demonstrated that both periodic and discrete motions can be embedded at the same time and with the ability to switch between them. The underlying dynamics of these motions are investigated as well. Finally, I proposed a dynamical system approach to obstacle avoidance that is inspired on vector fields in the presence of repellers. This method is able to modify any motion trajectory in real-time, without the need of knowing the entire trajectory beforehand.

Adaptive Feedback Controller (Chapter 4)

The design of the Inverse Modeling Adaptive (IMA) control framework is presented of which an ESN-based implementation is evaluated on several control tasks. Its model independence is demonstrated by applying it on control tasks, each with different dynamics. I investigate its convergence and stability properties, and compare its performance with specifically designed task-depending control techniques.

Control Hierarchies (Chapter 5)

In this chapter two control hierarchies are presented. The first hierarchy introduces the concept of a biologically inspired control approach that tries to represent rich motion skills by the modulation and combination of a limited and predefined

set of motion primitives. By some elementary experiments this modulation concept is evaluated. The second hierarchy, called Modular Architecture with Control Primitives (MACOP), takes another direction. Instead of predefining the motion primitives, only a number of untrained IMA controllers are designated to represent the primitives. By defining a high level description of how these primitives should cooperate, the actual control primitives emerge unsupervised. Thanks to such heuristic definition it is shown that the complexity of the task is divided over several less complex tasks. MACOP is evaluated on an inverse kinematic learning task for a both a simulated and real-world robot.

Motion Planning and Control (Chapter 6)

The use of the previously proposed IMA controller is extended to the control of underactuated systems by incorporating a sample-based planning algorithm. By learning a forward model of the task, it is shown that the dynamics can be explored to find a state space trajectory that solves the task. Given this trajectory, MACOP is used to learn the corresponding control command. It is revealed that this approach introduces feedback in such a way that the approach becomes robust against inaccurate state space paths or noise. Furthermore, the concept of a simulation-based control framework is presented which learns both the dynamics and control iteratively and simultaneously. Finally, I discuss how this simulation-based control framework resembles to the underlying mechanisms of Reinforcement Learning.

Conclusions and Future Perspectives (Chapter 7)

Finally, an overview of this dissertation is given together with the corresponding contributions and conclusions. I also present some interesting future research directions that build upon the work in this dissertation.

1.7 List of publications

Journal publications

1. Nyman, J., Caluwaerts, K., Waegeman, T. and Schrauwen, B. (2013). Wall fitting for autonomous UAV exploration and visualisation using visual SLAM. *Robotica*, ?:?–?. (under review)

2. wyffels, F., Li, J., Waegeman, T., Jaeger, H. and Schrauwen, B. (2013). Frequency modulation of large oscillatory neural networks. *Biological Cybernetics*, ?:?–?. (under review)

3. Waegeman, T., Hermans, M., and Schrauwen, B. (2013). Macop modular architecture with control primitives. *Frontiers in computational neuroscience*, 7., pages 13.

4. Waegeman, T., wyffels, F., and Schrauwen, B. (2012). Feedback control by online learning an inverse model. *IEEE Transactions on neural networks and learning systems*, 23(10):1637–1648.

Conference publications

1. Degrave, J., Burm, M., Waegeman, T., wyffels, F., and Schrauwen, B. (2013). Comparing Trotting and Turning Strategies on the Quadrupedal Oncilla Robot. *International Conference on Robotics and Biomimetics*.

2. Degrave, J., Van Cauwenbergh, R., wyffels, F., Waegeman, T., and Schrauwen, B. (2013). Terrain Classification for a Quadruped Robot. *International Conference on Machine Learning and Applications*.

3. Waegeman, T., wyffels, F., and Schrauwen, B. (2012). Towards a neural hierarchy of time scales for motor control. *From Animals to Animats 12*, pages 146–155.

4. Waegeman, T., Schrauwen, B. (2012). A discrete/rhythmic pattern generating rnn. In *20th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN-2012)*, pages 567–572. Ciaco-i6doc. com.

5. wyffels, F., Bruneel, K., Bertels, P., D'Haene, M., Heirman, W., Waegeman, T. (2012). A human-friendly way of programming robots. In *5th International Workshop on Human-Friendly Robotics (HFR-2012)*.

6. Degrave, J., wyffels, F., Waegeman, T., Kindermans, P.-J., Schrauwen, B. (2012). Applying morphological changes during the evolution of quadruped robots results in robust gaits.

7. Nyman, J., Caluwaerts, K., Waegeman, T., and Schrauwen, B. (2012). System modeling for active noise control with reservoir computing. In *9th IASTED International Conference on Signal Processing, Pattern Recognition, and Applications (SPPRA-2012)*, pages 162–167. ACTA Press.

8. Waegeman, T. and Schrauwen, B. (2011). Towards learning inverse kinematics with a neural network based tracking controller. In *Neural Information Processing*, pages 441–448. Springer.

9. wyffels, F., D'Haene, M., Waegeman, T., Caluwaerts, K., Nunes, C., and Schrauwen, B. (2010). Realization of a passive compliant robot dog. In

Biomedical Robotics and Biomechatronics (BioRob), 2010 3rd IEEE RAS and EMBS International Conference on, pages 882–886. IEEE.

10. Waegeman, T., Antonelo, E., wyffels, F., and Schrauwen, B. (2009). Modular reservoir computing networks for imitation learning of multiple robot behaviors. In *Computational Intelligence in Robotics and Automation (CIRA), 2009 IEEE International Symposium on*, pages 27–32. IEEE.

Other publications

1. Khansari-Zadeh, S. M., Seungsu, K., Rückert, E., Waegeman, T., Lemme, A., Reinhart, F., Ajallooeian, M., and Gay, S. (2010). D 4.1, reaching benchmark (v1) results.

2. wyffels, F., Waegeman, T., Schrauwen, B., and Jaeger, H. (2012). Technical report on hierarchical reservoir computing architectures.

3. Ajallooeian, M., Gay, S., Ijspeert, A., Khansari-Zadeh, M., Kim, S., Billard, A., Rückert, E., Neumann, G., Waegeman, T., wyffels, F., Schrauwen, B., Lemme, A., Reinhart, F., Rolf, M., Steil, J., Carbajal, J. P., Sumioka, H., Zhao, Q., and Kuppuswamy, N. (2010). Comparative evaluation of approaches in t.4.1-4.3 and working definition of adaptive module.

2
Reservoir Computing

As mentioned in the Introduction, using linear regression as a modeling approach has the advantage of being easy and fast. However, the achieved modeling performance strongly depends on the task. When there exists a linear mapping from the observations to the desired outcome, a linear model suffices. On the other hand, when there only exists a non-linear mapping, more advanced regressors are needed to produce a good result. Approaches like Gaussian Process Regression, Gaussian Mixture Regression or Support Vector Regression are often used in such cases. Although they have a stronger modeling ability, they are more difficult and slower to train. Another approach to solving non-linear modeling problems is to transform the observations in such a way that, after the transformation, they can be linearly combined to achieve the desired non-linear mapping between the observation and the outcome. In this case, the difficulty is shifted to finding the right non-linear transformation.

The concept of Reservoir Computing (RC) is based on the observation that linearly combining the states of a Recurrent Neural Network with certain algebraic properties suffices to achieve competitive performance on a wide variety of practical applications (linear and non-linear).

I start this chapter by describing an Echo State Network (ESN) which is one of the original incarnations of RC. Although I have only used ESN in this dissertation, the chapter is concluded with a brief overview of some of the other techniques belonging to the RC paradigm together with a short enumeration of some of the application fields.

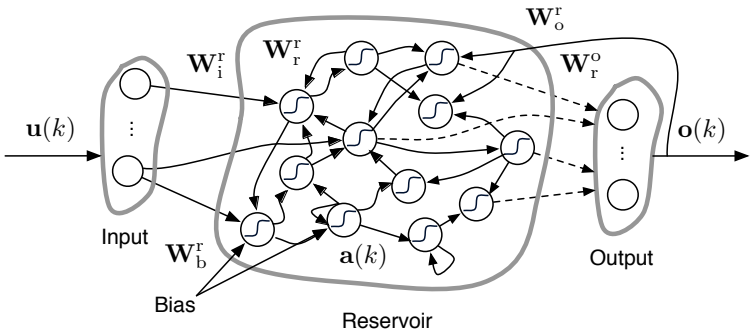


Figure 2.1: Description of an Echo State Network. Dashed arrows are the connections which can be trained. Solid arrows are fixed. $\mathbf{W}_g^h$ is a matrix representing the connections from g to h, where r,i,o,b denote *reservoir*, *input*, *output*, *bias*, respectively. $\mathbf{u}(k)$, $\mathbf{o}(k)$ and $\mathbf{a}(k)$ represent the input, output and neuron states, respectively.

2.1 Echo State Networks

2.1.1 General network topology

An Echo State Network consists of an input layer, a Recurrent Neural Network (RNN), referred to as reservoir, and an output layer (often called readout). The connections between these parts determine the topology of the ESN. There exist many possible topologies/hierarchies, each with their own interesting properties. However, the description in this section is limited to a topology which is used as a building block for the control approaches described in this book. As show in Figure 2.1, an ESN has connections running from the input layer to the reservoir and from the reservoir to the output layer. Furthermore, the use of output feedback connections is integrated in this topology description. Why and when these feedback connections are necessary will be described later on in this section.

2.1.2 Internal network state

One of the main purposes of using ESNs in this dissertation is to create a model of a plant. This requires that the timescale of the network dynamics matches to that of the plant dynamics. In this work most input signals driving an ESN are generated by plants, which are continuous dynamical systems. However, as we mentioned in the introduction, computer processing of such signals requires them to be discretized by sampling at a fixed sample rate. Choosing a sample rate that preserves

sufficient information in the input signal to create a good model is therefore very important. For instance, when the sample rate is too high, the intrinsic dynamics of an ESN with a neuron model described in Equation (1.2) might be too fast. To improve the modeling capability of an ESN, the underlying dynamics over a much longer time window need to be modeled as well. Down-sampling the input signal might reduce the length of the needed time window. It is possible, however, that essential information is lost when down-sampling the input signal. The sampling rate should be above the Niquist rate in order to get an alias-free signal sampling. Therefore, a more intelligent and continuous approach is needed. To slow down the ESN's dynamics, we take a look at a continuous time version of differential equation (Jaeger et al., 2007):

$$\dot{\mathbf{a}} = \frac{1}{c}\left(\tanh\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{a}(t) + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}(t) + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{o}(t) + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}\right) - \mathbf{a}(t)\right), \tag{2.1}$$

where $c > 0$ denotes a time scale global to the ESN and where $a_i(t)$ denotes the neuron state of the ith neuron of all N neurons in the reservoir. Furthermore, $\mathbf{u}(t)$ is the input signal and $\mathbf{o}(t)$ the output produced by the ESN with a dimensionality of respectively N_{in} and N_{out}. The feedback signal (from output to reservoir) has a dimensionality of N_{out} which allows the reservoir to use its own generated output signal $\mathbf{o}(k)$ as input. Additionally, an input and output bias are assumed. The input bias can be considered as an extra input dimension with a constant signal equal to one. The output bias can be considered as an extra reservoir state with a constant value equal to one. The weight matrices $\mathbf{W}_g^h$ represent the connections from g to h, where r,i,o,b denote *reservoir*, *input*, *output* and *bias*, respectively. The corresponding matrices $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$, $\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$, $\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$ and $\mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$ have a dimensionality of $N \times N$, $N \times N_{\mathrm{in}}$, $N \times N_{\mathrm{out}}$ and $N \times 1$, respectively. Throughout this work we use the $\tanh()$ function as a non-linearity that bounds the states $\mathbf{a}(t)$ to values between -1 and 1. An Euler discretization with a step size of Δt (assumed to be small) approximates the derivative of an activation function as follows:

$$\dot{\mathbf{a}} \approx \frac{\mathbf{a}(t+\Delta t) - \mathbf{a}(t)}{\Delta t}. \tag{2.2}$$

By introducing an extra parameter $\gamma = \frac{\Delta t}{c}$, called *leak rate*, the continuous time neuron dynamics are approximated by the following update equation:

$$\mathbf{a}(t+\Delta t) = \mathbf{a}(k+1) = (1-\gamma)\mathbf{a}(k) + \gamma\tanh\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{a}(k) + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}(k) + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{o}(k) + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}\right). \tag{2.3}$$

These neuron states are also called *echos* because they are non-linear transformations of the current and past inputs. The input signal $\mathbf{u}(k)$ is assumed to be a finite sequence of data. This is needed if we want to train the ESN in a finite amount of time. However, as I will discuss later on, it is possible to train the ESN incremen-

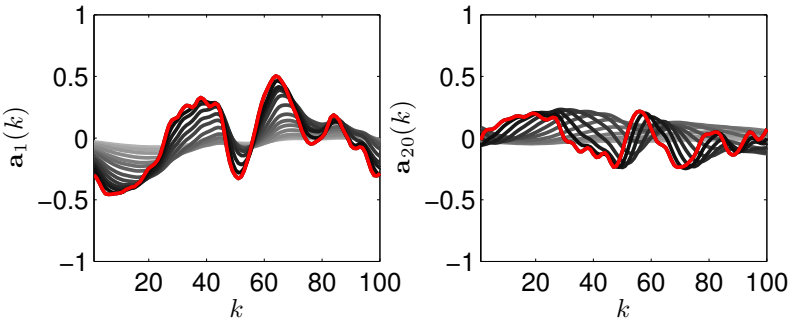


Figure 2.2: Both plots illustrate the influence of the leak rate parameter γ on two neuron states (left plot: 1st neuron, right plot: 20th neuron) over a course of 100 time samples. The gray scale of each signal indicates the choice of a γ from the range $[0.01, 0.0141, 0.0200, \cdots, 0.6310, 0.8913]$ from light gray to black. The red signal represents $\gamma = 1$ (no leak rate).

tally (or online) where new data is provided during operation of the ESN.

By tuning the leak rate, the dynamics of the network can be slowed down in such a way that they match the intrinsic timescale which is needed to model the plant. To illustrate the influence of this leak rate, I show in Figure 2.2 how the network state dynamics are affected by γ. It is interesting to note the temporal smoothing of the neuron state and its effect on the amplitude. Using no leak rate allows for high frequencies within the neuron states to persist. Although such a ESN has *fading memory*[1] due to its recurrent connections, this fading memory is often too small to hold the necessary modeling information. A low leak rate on the other hand, allows for the past state to have its effect on the current state yielding a smoother signal with less high frequencies but a larger fading memory.

2.1.3 Input connections

The connections between the input layer and the reservoir are represented by the $\mathbf{W}_{\text{i}}^{\text{r}}$ matrix with a size of $N \times N_{\text{in}}$. Each column contains weights drawn from a standard normal distribution $\mathcal{N}(0,1)$[2] and represent the connections from a single input dimension to all N neurons. When such a weight is equal to zero, the corresponding

[1] In order for the ESN concept to work, the reservoir is conditioned in order to obtain the Echo Sate Property (ESP). In Section 2.1.4.1 this will be explained in more detail. However, less formally, when the ESP is fulfilled, the reservoir has a fading memory in which any information from initial conditions diminishes asymptotically over time.

[2] Other choices are possible, e.g., $\mathcal{U}(-1,1)$.

neuron is unaffected by the value of this input dimension which is equivalent to not having a connection.

2.1.3.1 Input scaling

The non-linearity used in Equation 2.3 has an approximately linear and a very non-linear part. Which part is activated depends on the input signal $\mathbf{u}(t)$ and the input weights $\mathbf{W}_{\text{i}}^{\text{r}}$, among others. When they are too small, the ESN will have more linear dynamics and when they are larger more non-linear network dynamics will be present. However, making them too large causes the saturation of the $\tanh()$-function, in such a way that the neuron states become constant and prevent them to encode the needed dynamics for the task. The scaling of the input also depends on the number of input signals as more inputs sum to a larger value and thus can saturate the used non-linearity more easily. Therefore, normalizing the input signal and choosing an *input scaling factor* that regulates the global scaling is important. Throughout this work, unless mentioned otherwise, we will normalize the input signal as follows:

$$\mathbf{u}_n(k) = \frac{\mathbf{u}(k) - \mu}{\sigma}, \tag{2.4}$$

where μ and σ denote the mean and standard deviation of $\mathbf{u}_n(k)$ value's range. The weights $\mathbf{W}_{\text{i}}^{\text{r}}$ are scaled by the input scaling factor f_i^r which is equivalent to drawing the weights from the normal distribution $\mathcal{N}(0, f_i^r)$. As the output feedback weights $\mathbf{W}_{\text{o}}^{\text{r}}$ have the same purpose as the input weights, they are chosen from the distribution $\mathcal{N}(0, f_o^r)$ where f_o^r is called the *feedback scaling factor*. In Figure 2.3, two neuron states are shown for different input scaling factors. It is clear that there exists no temporal smoothing effect when modulating the input scaling. Furthermore, one can notice that the non-linear changes in shape are caused by the properties of the used hyperbolic tangent function (e.g., signal saturation near 1 and -1). The right plot of Figure 2.3 shows an unexpected scaling behavior in the neuron state compared to the left plot. When we neglect input scaling (red signal) the impact of the previous neuron state is longer than the current input. However, by increasing the input scaling the input becomes more dominant and as a result the specific neuron is affected more by the larger neuron activity of the other neurons and its own input, compared to its own state. This also depends on the weights between the neurons and with what magnitude they can change the neuron state.

2.1.3.2 Bias

In Equation 2.3 we use a $\tanh()$-function as non-linearity. This is an odd (antisymmetric) function:

$$\tanh(-x) = -\tanh(x). \tag{2.5}$$

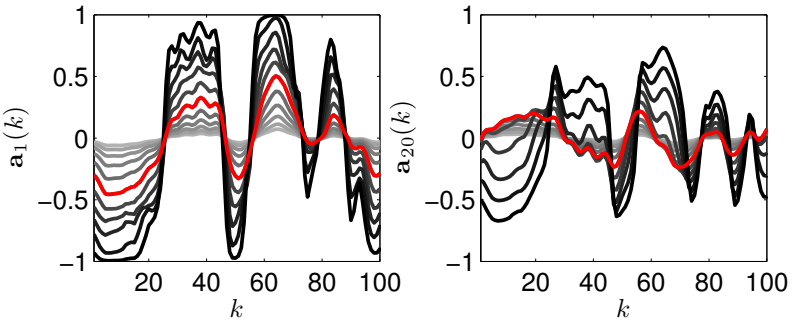


Figure 2.3: Both plots demonstrate how 2 neuron states change by the input scalings factor f_i^r over a course of 100 time samples. The gray scale of each signal indicates the choice of a f_i^r from the range $[0.1, 0.1585, 0.2512, \cdots, 6.3096, 10]$ from light gray to black. The red signal represents $f_i^r = 1$ (the input is not globally scaled over all input weights).

Although for some tasks an antisymmetric function as non-linearity might be sufficient, for most tasks it is needed to use a non-antisymmetric non-linearity. Introducing an extra *input bias* parameter breaks this antisymmetry by shifting the origin upwards or downwards depending on its value. Furthermore, the Taylor expansion of the tanh function consists of odd powers, and introducing a bias term causes the occurrence of even powers in its expansion. The introduced input bias can be considered as an extra input dimension with a constant value f_b^r. In Equation 2.3 the input bias is denoted by $\mathbf{W}_{\text{b}}^{\text{r}}$ of dimensionality $N \times 1$ where its elements are drawn from the $\mathcal{N}(0, f_b^r)$ normal distribution. In Figure 2.4, two neuron states are shown for different bias values. The red signal represents the neuron state without bias. When bias is added increasingly, the neuron state signal changes from black to light gray.

2.1.4 Recurrent network connections

As stated in Section 2.1.1, an ESN contains a Recurrent Neural Network, called reservoir, which is represented by a square connection matrix $\mathbf{W}_{\text{r}}^{\text{r}}$. Each element $\mathbf{W}_{\text{r},ij}^{\text{r}}$ represents the connection weight from the i-th to the j-th neuron. In this work the connection weights are drawn from a standard normal distribution $\mathcal{N}(0,1)$. However, other distribution choices are also possible (e.g., uniform distribution $\mathcal{U}(-1,1)$).

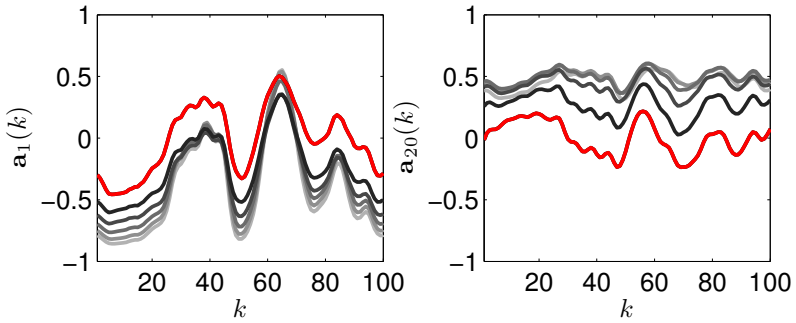


Figure 2.4: In this graph we demonstrate how 2 neuron states are influenced by the use of a bias constant f_b^r where t represents the time in number of samples. The choice of a f_b^r from the range $[1, 0.8, \cdots, 0.2, 0]$ is represented by the gray scale of each signal from light gray to black, respectively. The red signal represents $f_b^r = 0$ (when no bias is used).

2.1.4.1 Spectral radius

Together with the bias terms, the recurrent connection weights mainly define the dynamical regime in which the network operates. In order for ESNs to useful, these recurrent connections should obey the *echo state property*. As described in Lukosevicius and Jaeger (2009) this property states that the effect of the current input $\mathbf{u}(k)$ and reservoir state $\mathbf{a}(k)$ on the future state $\mathbf{a}(k+j)$ should gradually diminish over time ($j \to \infty$). Furthermore, the influence should not persist or get amplified. Because of the used non-linearity it is quite difficult to investigate the influence of the chosen connection weights on the dynamical regime of the network. However, by linearly approximating the $\tanh()$ non-linearity, the reservoir state can be considered stable if all the eigenvalues of $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ are smaller than or equal to one. If there exist eigenvalues that are larger than one, the neuron state will start to grow exponentially until they get saturated by the used non-linearity. To prevent this from happening, the recurrent connection weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ are scaled by the largest absolute eigenvalue so that the largest absolute eigenvalue becomes equal to one. Next, by scaling all weights with a parameter ρ called *spectral radius*, the dynamic regime of the network can be regulated. Although for most practical applications the echo state property is satisfied when $\rho < 1$, this is not a necessary nor sufficient condition for the presence of the echo state property and should therefore rather be used as a rule of thumb. As stated in Verstraeten et al. (2007); Lukosevicius and Jaeger (2009); Caluwaerts et al. (2013b); Yildiz et al. (2012), the echo state property can still hold for $\rho > 1$ with non-zero inputs and maybe lost for smaller networks even when $\rho < 1$. In Figure 2.5 two neuron states are shown for several

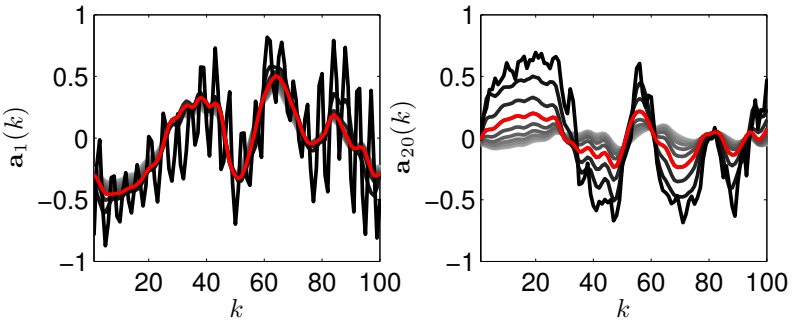


Figure 2.5: In this graph we demonstrate how 2 neuron states change for different choices of spectral radius ρ where t represents the time in number of samples. The choice of a ρ from the range $[0.5, 0.5610, 0.6295, \cdots, 1.2560, 1.4093]$ is represented by the gray scale of each signal from light gray to black, respectively. The red signal represents $\rho = 0.99$ (default spectral radius).

of ρ choices. A large spectral radius causes high aperiodic frequency changes in the neuron state which is a typical observation of chaotic behavior. The lower the spectral radius the more damped the system becomes.

2.1.4.2 Connection fraction

As mentioned in Section 2.1.2 each element $\mathbf{W}^{\mathrm{r}}_{\mathrm{r},ij}$ of $\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}$ represents the weight of the connection from neuron i to neuron j. When this weight is equal to zero, there exists no connection between both neurons. The sparser the matrix $\mathbf{W}^{\mathrm{r}}_{\mathrm{r},ij}$ becomes, neurons become less and less connected to the other neurons. The fraction of neurons a single neuron is connected to is regulated/indicated by the *connection fraction*. When the connection fraction is 100%, each neuron is connected to itself and to all other neurons. When a lower connection fraction is employed, randomly distributed connection weights will be zeros. One of the reasons researchers use sparse weight matrices, is because the execution of the computations can be performed much faster. For instance in Triefenbach et al. (2010), the use of a sparse connection matrix allowed the researchers to use ESNs of up to $20,000$ neurons. It has been shown in Büsing et al. (2010) and Drossel (2008) that the connection fraction in non-linear networks with saturated neurons and neurons with a threshold function as non-linearity, respectively, have a large influence on the network's performance. However, as long as the neurons operate in a quasi linear regime, the connection fraction has little to no influence. This experience matches with the experiments performed in Sun et al. (2012); Strauss et al. (2012); wyffels (2013) that

show that the echo state property of the network is more essential than the topology of the connections. Because in this work the ESNs are rather small in size, and computational advantages are limited, I always use fully connected networks.

2.1.4.3 Linear memory capacity

The reservoir of an ESN is actually a high dimensional dynamical system and as the word 'dynamic' suggests that its current state depends on a fading history of previous reservoir states. In other words, the dynamical system has a certain memory of its previous states and also of the inputs which influence these states. If we want to consider the use of ESNs for modeling/control applications, we need to be able to quantify the amount of memory a reservoir has, and how this *memory capacity* is affected by the parameters of the reservoir. Consider an ESN with a single input $u(k)$ and output $o(k)$ that is trained to produce a delayed version of its input as output ($o(k) = u(k-\delta)$). When only the linear part of the used non-linearity is considered one can arguably consider, the linear memory capacity of this ESN as a rough indicator of how long the history of the input signal is remembered by the ESN. In order to measure the ESN's ability to remember the history of the input, the correlation between the input and the output signal is used. This correlation can be written more formally as:

$$C(\delta) = \frac{\mathrm{cov}(u(k-\delta), o(k))}{\sigma_u \sigma_o}, \tag{2.6}$$

with σ_u and σ_o the standard deviation of the input and output signal, respectively. When there is no correlation for a delay δ, $C(\delta) = 0$. Note that the applied input signal should be uncorrelated noise because we only want to measure the memory properties of the reservoir and any correlation within the input signal itself will clutter these measurements. If we take the square of $C(\delta)$, any correlation will be indicated by a value that is bigger than zero but smaller than or equal to one. The resulting function $m(\delta) = (C(\delta))^2$ is called the *memory function*. Furthermore, the *linear memory capacity M* is defined as:

$$M = \sum_{\delta=0}^{\infty} m(\delta). \tag{2.7}$$

The memory capacity of the reservoir is influenced by the properties which cause the dynamical system to have a memory in the first place. For instance, the use of recurrent connections allows a neuron state to depend on its previous state and on that of other neurons. Additionally, we showed before that the use of leaky integrator neurons modulates the temporal dependency of a reservoir state to its history. In Figure 2.6 the memory function is plotted for different leak rates and input scalings in the left and right plot, respectively. For every experiment 50 neurons were

used. The red curve represents the use of no leak rate ($\gamma = 1$). As γ decreases in value from dark to light gray, the tail of the memory function becomes increasingly longer. The impact of the input scaling upon the memory function is more dramatical. Lowering the value of the input scaling (from dark to light gray) causes the reservoir to behave more linearly (linear region of hyperbolic tangent), yielding a bigger area under the corresponding memory curve and thus a larger memory capacity. There exists of course an upper limit to the amount of memory a ESN can have. As been suggested by Hermans (2012), one can encode at most N numbers from a set of N uncorrelated other numbers without loosing precision. This means, that the memory capacity is at most equal to N. Therefore, increasing the reservoir size also increases the potential memory capacity of the ESN. In Figure 2.7 several memory functions are shown, each for a different spectral radius. In the left plot the influence of the spectral radius is less clear because of the non-linear behavior of the network. However, by lowering the input scaling $f_i^r = 0.001$ the network becomes more linear and, as shown in the middle plot, the impact of the spectral radius on the memory function becomes more clear. In both the left and middle plot, the spectral radius ρ changes from 0.5 (light gray) to 1.5 (black) or from linear to possible chaotic behavior. The red curve represents $\rho = 0.99$. It is interesting to note that in the middle plot the memory capacity increases for a larger ρ as long as it is smaller or equal to one. When the spectral radius becomes larger than one, the memory capacity rapidly decreases, although the network still retains some past information. The right plot of Figure 2.7 demonstrates how the reservoir size affects the corresponding memory function. The light gray and black curves represent 20 and 80 neurons, respectively, while the red curve corresponds to a reservoir of 50 neurons. Again to clarify the influence $f_i^r = 0.001$ is used. The main drawback of analyzing linear memory capacity, is that it ignores any non-linear transformation of the input signal. An interesting approach to include non-linear transformations into the analysis of memory capacity, is by additionally investigating how well non-linear functions of the input signal can be reproduced (Dambre et al., 2012). In this work, however, only linear memory capacity is considered.

2.1.5 Output connections

The output connections are the connections between the reservoir and the output layer, and are represented by the weight matrix $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$ with dimension $N + 1 \times N_{\mathrm{out}}$. These weights are the only weights in the ESN that are modified to achieve the desired network behavior. Modifying these weights is called training and in this section we will describe an *offline* and an *online* approach to train them. Both approaches are linear in the sense that they try to find a linear combination of features given by the ESN's reservoir states and an *output bias* term. By extending the reservoir states with a constant value, the value of this bias term is trained as well. Each

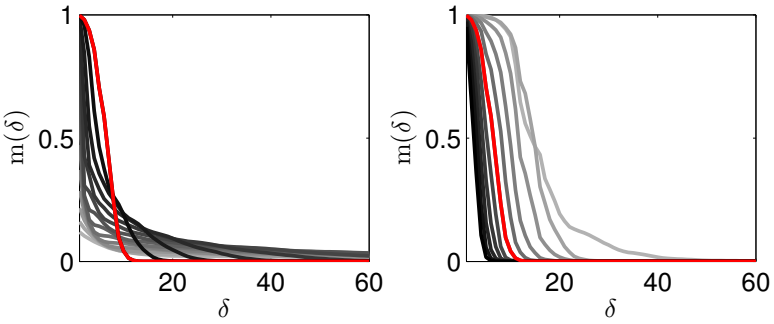


Figure 2.6: The left plot illustrate the influence of the leak rate parameter γ on the memory capacity. The gray scale of each curve indicates the choice of a γ from the range $[0.01, 0.0141, 0.0200, \cdots, 0.6310, 0.8913]$ from light gray to black. The right plot demonstrate how the memory function changes by the input scalings factor f_i^r. Here, the gray scale of each signal indicates the choice of a f_i^r from the range $[0.001, 0.0016, 0.0025, \cdots, 6.3096, 10]$ from light gray to black. In both plots the red curve represents $\gamma = 1$ and $f_i^r = 1$ (no leak rate and input scaling).

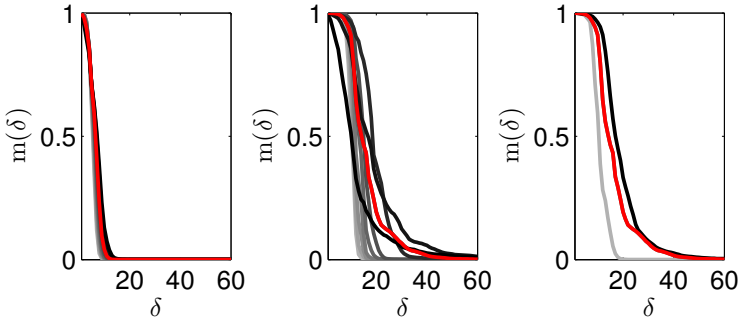


Figure 2.7: The left and middle plot illustrate the impact of the spectral radius upon the memory capacity. However, the left and middle plot differ in the used input scaling factor (left plot $f_i^r = 1$, middle plot $f_i^r = 0.001$). The choice of a ρ from the range $[0.5, 0.5610, 0.6295, \cdots, 1.2560, 1.4093]$ is represented by the gray scale of each curve from light gray to black. The red curve in both plots represents $\rho = 0.99$. The right plot demonstrates the effect of the reservoir size on the memory capacity. The gray, red and black line denote a reservoir size of 20, 50 and 80 neurons, respectively.

column in the *design matrix* $\mathbf{\Phi}$ is a concatenation of all used features at time k:

$$\mathbf{\Phi} = \begin{bmatrix} \mathbf{a}(0) \ldots \mathbf{a}(k) \ldots \mathbf{a}(K) \\ 1 \quad \ldots \quad 1 \quad \ldots \quad 1 \end{bmatrix} \tag{2.8}$$

Given this design matrix, the ESN output $\mathbf{o}(k)$ is defined by each column in the following matrix:

$$\mathbf{O} = \mathbf{W}_{\mathrm{r}}^{\mathrm{o}T} \mathbf{\Phi}. \tag{2.9}$$

2.1.5.1 Output feedback

Work by Jaeger and Haas (2004), has demonstrated that ESN perform well in predicting time series. In order to simplify this task, the ESN's output is fed back to the reservoir. When output feedback is present in the ESN, the feedback signal during training does not correspond to the actual produced output but is clamped to the target output. Therefore, this procedure has the appropriate name *teacher forcing*. To improve robustness and generalization capabilities a small amount of noise is added to this teacher forced signal. I would like to refer to wyffels (2013), for a more in depth study of Reservoir Computing systems with output feedback

In the following sections, I will present two different procedures for training the output connection weights.

2.1.5.2 Linear regression

In order to train the output weights in such a way that the network output $\mathbf{o}(k)$ corresponds to a target output $\mathbf{o}_{\text{target}}(k)$, a quadratic error is typically minimized. Often the *Mean Square Error* (MSE) is used as an error metric to define the quadratic difference between the network output and its target signal averaged over time. More formally:

$$\text{MSE} = \frac{1}{K} \sum_{k=1}^{K} \|\mathbf{o}(k) - \mathbf{o}_{\text{target}}(k)\|^2 \tag{2.10}$$

$$= \frac{1}{K} \|\mathbf{W}_{\mathrm{r}}^{\mathrm{o}T} \mathbf{\Phi} - \mathbf{O}_{\text{target}}\|_F, \tag{2.11}$$

with $\|\cdot\|_F$ the Frobenius norm. Minimizing the MSE with respect to the output weights is achieved by linear regression:

$$\mathbf{W}_{\mathrm{r}}^{\mathrm{o}} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} (\mathbf{\Phi}^T \mathbf{O}_{\text{target}}). \tag{2.12}$$

Calculating $(\mathbf{\Phi}^T \mathbf{\Phi})^{-1}$ to solve a system of linear equations explicitly is computationally demanding and does not give a good numerical solution when there exists no unique solution for this system. Therefore, the Moore-Penrose pseudoinverse

(Albert, 1972) $\mathbf{\Phi}^{+} = (\mathbf{\Phi}^{T}\mathbf{\Phi})^{-1}\mathbf{\Phi}^{T}$ is often applied to compute a numerically stable least squares solution for the system of linear equations even if $\mathbf{\Phi}^{T}\mathbf{\Phi}$ is ill conditioned.

2.1.5.3 Regularization

Training an ESN to approximate an unknown relationship between certain observations and a corresponding output has the advantage that the underlying processes need not be known explicitly to achieve a good model. Instead, based on a limited set of acquired (observations, outcome) examples, an ESN implements the model implicitly. On the other hand, the disadvantage of such an approach is that the model is based on a limited number of examples, resulting in a model that might be very accurate for the training examples but really bad when it needs to generalize to unseen observations. Often, this poor generalization is related to the modeling power of the learning approach. For instance, fitting a simple linear model to some noisy data will generalize better than a model that uses all data points and in a sense has learned all examples by heart instead of finding an underlying relationship between the model input and output.

This problem of fitting the data too well is called *overfitting*. To prevent the model from being to sensitive to noise and thus overfitting the data, a trade-off has to be made between modeling complexity and generalization ability. The modeling power of an ESN is largely influenced by the size of its reservoir. Therefore, in the past, the size of the reservoir was constrained in order that it was not too large or too small, avoiding it to overfit or underfit the model, respectively. Optimizing the reservoir size is computationally costly. Another approach to prevent *overfitting* is to regularize the network by imposing some other constrains on the model (e.g., some smoothness constraints). Often, normally distributed noise is added to each neuron state during training so that the network becomes more robust to small variations in the neuron state. As a result, the ESN becomes more robust against small variations in the input, which allows the network to generalize better. A more commonly used approach of regularization is to impose constrains on the norm of the trained connection weights in such a way that they can not become too large. More formally, the cost function $\mathcal{J}$ that we try to minimize becomes:

$$\mathcal{J}(\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}) = \mathrm{MSE} + \nu\|\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}\|_F \tag{2.13}$$

$$= \|\mathbf{W}_{\mathrm{r}}^{\mathrm{o}T}\mathbf{\Phi} - \mathbf{O}_{\mathrm{target}}\|_F + \nu\|\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}\|_F, \tag{2.14}$$

with $\|\cdot\|_F$ the Frobenius norm and ν the *regularization parameter*, which regulates the trade-off between the norm of the trained connection weights and the error on the training data. Minimizing the cost function above is also known as *ridge regression* or Tikhonov regression (Tikhonov and Arsenin, 1979; wyffels et al., 2008). Given the cost function above, the resulting output weights can be calculated

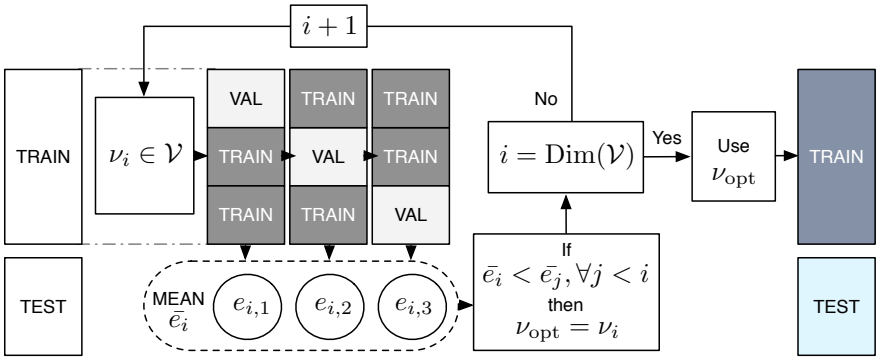


Figure 2.8: Illustration of the K-fold cross-validation process for $K = 3$. $\mathcal{V}$ denotes a predefined set of possible regularization parameters ν_i. For each configuration k of validation (defined as 'VAL') and train set the validation error $e_{i,k}$ is determined by calculating the MSE. The average error $\bar{e}_i$ over all K configurations is compared to the average validation error of the other regularization parameters. If ν_i is smaller than the other options, ν_i becomes the optimal regularization parameter ν_{opt}. When all possible regularization parameters are picked ν_{opt} is used to train the network with all available train data and is consecutively evaluated on the test data (not used before).

in one shot by modifying Equation (2.12) to:

$$\mathbf{W}_{\mathrm{r}}^{\mathrm{o}} = (\mathbf{\Phi}^T\mathbf{\Phi} + \nu\mathbf{I})^{-1}(\mathbf{\Phi}^T\mathbf{O}_{\mathrm{target}}), \tag{2.15}$$

where $\mathbf{I}$ denotes the identity matrix. The regularization parameter ν needs to be optimized for the used ESN and training data so that it optimally generalizes to unseen data. Commonly, a grid-search is employed to find parameters that perform optimally for a task. I will explain this optimization process for the case of the regularization parameter although it is applicable for any ESN-parameter. During grid-search every value among a set of possible regularization parameter values is picked and used to train an ESN using only a part of the data (the training set). Its performance is validated on an unseen part of the data. To remove the chance of choosing the training and validation set badly, K-fold *cross-validation* is applied. As shown in Figure 2.8, the available data for training is divided into K equally sized folds/parts of which $K-1$ parts are used for training the ESN, and 1 part for evaluating the performance of the chosen ν. Next, the process is repeated K times until all folds are used once for validation for a given ν. The average performance

of the chosen ν on all the validation folds determines when the chosen parameter is optimal compared to the others. After finding an optimal regularization parameter, the ESN is trained again, but on all parts used in the K-fold cross-validation and tested on yet another unseen part of the data (the test set).

2.1.5.4 RLS and FORCE-learning

Equation 2.15 allows us to train the output weights $\mathbf{W}_{\text{r}}^{\text{o}}$ in one shot when all the data is known. Such an approach is called an *offline learning* method. In a task where new data becomes available during the task (*online learning*), it is possible to add this new data and calculate the output weights again. However, the covariance matrix need to be recalculated as well, which becomes more and more computationally intensive.

RECURSIVE LEAST SQUARES

Other approaches such as Recursive Least Squares (RLS) (Plackett, 1950) approximate the inverse of the covariance matrix iteratively. As a result, such an incremental approach can adjusts the output weights $\mathbf{W}_{\text{r}}^{\text{o}}$ every time new data is provided without having the burden of becoming slower as more data becomes known.

Similar to the described offline linear regression approach, the error we want to minimize is the difference between the predicted model output $\mathbf{o}(k)$ and the target output $\mathbf{o}_{\text{target}}(k)$:

$$\begin{aligned}\mathbf{e}(k) &= \mathbf{o}(k) - \mathbf{o}_{\text{target}}(k) && (2.16)\\ &= \mathbf{W}_{\text{r}}^{\text{o}}(k-1)^T\mathbf{\Phi}(k) - \mathbf{o}_{\text{target}}(k). && (2.17)\end{aligned}$$

The running estimate of the inverted covariance matrix $(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}$ is calculated as follows:

$$\mathbf{P}(k) = \frac{\mathbf{P}(k-1)}{\lambda} - \frac{\mathbf{P}(k-1)\mathbf{\Phi}(k)\mathbf{\Phi}^T(k)\mathbf{P}(k-1)}{\lambda(\lambda + \mathbf{\Phi}^T(k)\mathbf{P}(k-1)\mathbf{\Phi}(k))}. \tag{2.18}$$

where λ denotes the *forgetting factor* that controls the rate at which the influence of historical data diminishes. Choosing λ depends on the task at hand. The smaller λ is, the smaller the contribution of previous data points and the larger the changes to the output weights. If λ becomes too small, the RLS algorithm can become unstable. In contrast, if $\lambda = 1$, the algorithm will take all data points into account equally. Commonly, λ is chosen between 0.98 and 1 (Ifeachor and Jervis, 2002). Whenever I apply RLS, I use $\lambda = 1$ unless mentioned differently. After calculating both equations the output weights are updated such that they take the newly provided data into account:

$$\mathbf{W}_{\text{r}}^{\text{o}}(k) = \mathbf{W}_{\text{r}}^{\text{o}}(k-1) - \mathbf{e}(k)\mathbf{P}(k)\mathbf{\Phi}(k). \tag{2.19}$$

For both update Equations (2.18) and (2.19) the initial conditions are set to $\mathbf{P}(0) = \frac{\mathbf{I}}{\alpha}$ and $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}(0) = \mathcal{N}(0,1)$, respectively. Here, the parameter α acts as a learning rate and should be tuned depending on the particular task. However, in practice, α is chosen arbitrarily but < 1. These initial conditions are called soft constraints in literature. RLS is equivalent to ridge regression if the forgetting factor $\lambda = 1$ (Ismail and Principe, 1996). Therefore, α is sometimes referred to as a regularization parameter instead of a learning rate.

RLS is very similar to the Kalman filter which is in fact a generalization of RLS. When the variance of the process and measurement noise are equal to respectively zero and one, the Kalman filter is equivalent to RLS with $\lambda = 1$.

The weight updates calculated by RLS are initially very large for a few algorithm iterations after which the magnitude of the weight changes gradually becomes smaller. In other words, RLS tries to minimize $\mathbf{e}(k)$ as fast as possible by making rapid and effective weight modifications.

FORCE LEARNING

This property of minimizing the error as fast as possible, makes RLS suitable for the FORCE (first-order reduced and controlled error) learning procedure proposed by Sussillo and Abbott (2009). The FORCE learning approach modifies connection weights either external to or within a recurrent neural network so that chaotic spontaneous activity (spectral radius $\rho > 1$) is changed into a wide variety of desired activity patterns. This is in contrast to standard ESN learning approaches where the network activity ought to be non-chaotic (rule of thumb $\rho < 1$). Furthermore, in FORCE learning, the feedback loop from the output layer to the network are left intact and unclamped during training. Even if the produced output is incorrect during the first few algorithm iterations, the algorithm is robust enough to handle such differences together with the chaotic network activity. The FORCE procedure closely approximates the desired target signal but it does not clamp it. These small numerical differences between the actual and target network output during training allows the approach to sample instabilities in the recurrent neural network and stabilize them. Another interesting observation made by Sussillo and Abbott (2009) is that the magnitude of the trained weights becomes smaller if the chosen spectral radius of the network is larger. One can thus argue that the spontaneous and chaotic network activity is needed for regularization purposes to achieve robustness and generalization capabilities. Additionally, the number of trials needed to train the network is lower when the spectral radius is larger. Given these observations, one could consider to choose the spectral radius around $\rho = 1.5$. However, it depends on the speed at which the training information is provided in the form of training examples. An algorithm that learns too fast will over-fit on the limited data seen during training. For control tasks, where useful training data becomes available at a slower rate than when trying to learn a simple pattern generator, it is important not to choose a spectral radius that is too large.

2.2 Other Reservoir Computing flavors

I mentioned before that around 2001 the ESN approach was proposed to tackle the several shortcomings of which classical training approaches of Recurrent Neural Networks suffer. Around the same time Wolfgang Maas independently proposed a similar approach, called Liquid State Machines (LSM) (Maass et al., 2002), with the same conceptual idea of only training the readout but for spiking neurons instead of analog neurons. Both ESN and LSM had predecessors in computational neuroscience (Dominey, 1995) and subsequent branches in machine learning such as Backpropagation-Decorrelation by Steil (2004). They are now commonly known as Reservoir Computing approaches. Although in this work, all modeling problems are solved by using ESNs, we briefly describe some of the other RC flavors.

2.2.1 Liquid State Machines

LSMs originate from a computational neuroscience background where their purpose was to clarify principal computational properties of neural microcircuits. The internal dynamical system, referred to as *liquid*, follows the analogy of the excited network state as being ripples on a water surface. Instead of using simple sigmoid neurons, more sophisticated spiking neuron models are used. This approximates the physical brain neuron and its dynamical synaptic connections more accurately. Furthermore, the internal topology of the recurrent neural network is more restricted resulting in a more realistic from a biological point of view. The internal connections are randomly chosen within these biological constraints and remain fixed. All training occurs in the output connections. As a result, LSMs can perform more complicated information processing and they are more suited than ESNs to transfers natural neural mechanisms to. However, this comes at the cost of being harder to implement, to tune and to emulate. All this added complexity compared to ESNs makes them less popular for engineering applications.

2.2.2 Backpropagation-Decorellation

Backpropagation-Decorellation (BPDC) is a learning rule that resembles the idea of separating the reservoir and readout and only train the latter. However, BPDC is based on an analysis of the weight dynamics of a RNN trained with a special error gradient method (Atiya and Parlos, 2000). Steil (2004) observed that the output weights were changing quickly compared to the rate at which the internal weights were changing. BPDC exploits this observation by keeping the RNN weights fixed after being chosen randomly and only training the output weights in an iterative and online fashion.

2.2.3 Other types of reservoirs

The reservoir within a Reservoir Computing system does not necessarily have to be a recurrent neural network. As mentioned previously, an RNN behaves like a dynamical system. Therefore, other high dimensional dynamical systems with a fully or partially observable state that can be influenced by an external input, can be used as well. For instance, Fernando and Sojakka (2003) took the word 'reservoir' or 'liquid' quite literally and used a bucket of water as a replacement for the reservoir. They perturbed the water with mechanical actuators and used the observed surface ripples as a feature vector. Based on these features, a simple readout perceptron was trained to solve the XOR problem and to undertake speech recognition. Temporal patterns of excitation are converted by the water to observable spatial patterns. To clarify this, one can think about 2 consecutive drops perturbing a water surface in the middle of a bucket. The ripple effect of the first drop followed by the second drop allows the difference in time between both drops to be observable at any point in time after the drops have touched the water and before the ripples have faded away. Next to the inherent stability of water, local interference between the ripples caused by simultaneous sensory inputs allows for non-linear parallel computing.

The idea of morphological computation, where the properties of the body are exploited to offload computations needed for locomotion (or other processes), so that control is simplified, has been linked to Reservoir Computing as well. Here, the body acts as the observable dynamical system, i.e., the reservoir. Researchers have investigated how abstract representations (e.g., mass-spring-damper systems) of a biological system can be used to do computations (Caluwaerts and Schrauwen, 2011; Hauser et al., 2012; Caluwaerts et al., 2013a). Instead of using a structurally different physical representation of a reservoir, Vandoorne et al. (2008) made a photonic implementation of a RNN in which each neuron is represented by a Semiconductor Optical Amplifier (SOA). Although the behavior of a single neuron differs drastically from a hyperbolic tangent neuron, the researchers have demonstrated that they can successfully learn classification tasks. Whether the use of an optical SOA network is beneficial in terms of power consumption and processing speed, however, remains to be seen. Other hardware implementations of a reservoir use analog computing devices, such as a VLSI neural network, with which they can achieve similar results as when using a software RC implementation (Schürmann et al., 2004). Surely, the concept of RC can be transferred to a wide variety of fields in which its use still needs to be discovered.

2.3 Applications

ESN and Reservoir Computing (RC) approaches in general have many applications in a wide variety of fields. I would like to conclude this chapter by giving a brief overview of some of these applications and their results.

- **Time series prediction:** In Jaeger and Haas (2004) an RC approach is used to predict the behavior of chaotic systems such as the Mackey-Glass and Lorenz attractors (Mackey and Glass, 1977; Lorenz, 1963), in both cases setting a new state-of-the-art performance. Here the RC approach was trained to predict the system state one time step ahead. However, by using the trained approach recursively, it becomes possible to predict more than one step ahead. Consequently, for predicting the weather or stock market behavior such an approach becomes really attractive (wyffels and Schrauwen, 2010).

- **Epileptic seizure detection:** The good modeling abilities of RC approaches allow, their use in medical signal processing applications such as the detection of epileptic seizures in animal and human EEG data where they can compete with state-of-the-art solutions (Buteneers et al., 2012).

- **Speech recognition:** A more advanced application that exploits RC's modeling abilities concerns the classification of speech signals. In Verstraeten et al. (2006) for instance, the at the time state-of-the-art Hidden Markov Model (HMM) based recognizer was outperformed by a Liquid State Machine implementation. Later on, researchers from the Speech Lab in Ghent university proposed a large scale RC based hierarchy that is able to recognize the elementary sounds of speech (the so-called phonemes) with state-of-the-art performance while still leaving room for improvements (Triefenbach et al., 2013).

- **Robotic applications:** Another application field that is more related to the topic of this dissertation is that of robotics. In Reinhart (2011), a number of Back-Propagation De Correlation based approaches are applied in several robotics applications. The authors demonstrated the approach's ability to learn the inverse kinematics of a robot arm. In Antonelo (2011), ESNs are used to model a navigation system for autonomous mobile robots.

2.4 Conclusions

In this chapter, the Echo State Network (ESN) approach was presented, which is a particular Reservoir Computing approach. An ESN consists of an input layer, a randomly connected Recurrent Neural Network (RNN) and an output layer. Algorithms that train a RNN adapt all connection weights, which is a slow and tedious procedure of which the convergence can not be guaranteed. Furthermore, training all connection weights makes the learning process vulnerable to become disturbed by bifurcations. Instead of training all connections of the RNN, the ESN approach only trains the connection weights to the output layer. As a result, ESNs are easy to implement and the training can be achieved in one shot. The randomly connected RNN behaves as a dynamical system and, apparently, random dynamical systems are good non-linear dynamical models that outperform other methods on a number of tasks, e.g., time series prediction. ESNs and RNNs in general, are able to create a state representation of the time context in the data. It is also possible to use a Time Delayed Neural Network (TDNN) for this purpose. However, its modeling capacity is limited due to the fixed size of the time window (Trebatickỳ, 2009). Furthermore, TDNNs require a large number of parameters, especially when a large number of input/output dimensions are used (Ozturk, 2007). Based on these advantages, I will use ESNs to learn non-linear dynamical models.

3
Designing a Motion Pattern Generator (MPG)

In the introduction I mentioned that traces of self-operating machines, including artificial people, can be found in ancient mythologies. Engineers and inventors from ancient civilizations such as ancient China, Greece and Egypt tried to build such automated machines (robots), some of which resembled humans and animals. We might never know what the early motivations were for building such robots. A possible motivation might be that humans (biological systems) have an urge to minimize their labor by optimizing certain tasks. Arguably, this urge is rooted in evolution itself, where natural selection causes species to evolve in such a way that survival becomes possible.

Today robots can be found in most factories, increasing the accuracy and production speed of many products while reducing the labor. These robots operate in a fully structured and closed environment where the presence of workers is avoided. During the last century, however, researchers began to investigate how robots can be introduced in unstructured (non-deterministic) environments where human-robot interaction is possible. In such environments the complexity of the required control increases dramatically, making the use of classical programming approaches much more difficult.

Another more popular approach is that of *imitation learning* or *learning by example*, where the rich motion/motor skills of for instance humans are transferred to robots. Robots performing such movements are often perceived as being realistic and skillful, and are commonly used to bring robotic toys to 'life'. However, evaluation of the accuracy of these movements shows that they are not adaptive in any way. They are in fact recorded open loop movements; even when one records a large set of movements for a wide variety of situations, it is impossible to anticipate for all possible scenarios. To solve this problem, the recorded motions need to be extended to a more continuous space to allow generalization beyond the predefined context. When such a generalization exists, the robot can be connected in a closed loop yielding modified movements depending on the interaction with the

environment.

In this chapter I will describe the design of a motion pattern generator (MPG) that is able to generalize beyond demonstrated motion examples. This chapter starts with a short introduction to dynamical systems and some of their related properties, because these properties are important for understanding the underlying mechanisms of the presented MPG. Afterwards, the presented MPG is applied and evaluated on a simple robotic task and the generalization, robustness and modulating abilities are demonstrated on a set of recorded motions. Finally, I present a method of modifying the dynamics of the MPG so that obstacles can be avoided in real time while the motions are generated.

3.1 Dynamical system (DS)

Everything around us evolves over time. Because of the different magnitudes or time scales at which such changes occur, some of these time-dependent transformations are more apparent than others. For instance, when observing a plant from a macroscopic point of view, changes are visible at a slower time scale than when looking at the molecular (or even the cellular) level. Since the conceptual introduction of describing the motion of a physical object with the equations of Sir Isaac Newton, we have tried to mathematically formalize how everything around us evolves. A fixed formulation that describes how one or more variables evolves over time is said to describe a *dynamical system* and the variables are called the system's state. For instance, if we want to formalize the trajectory of a rotational pendulum, the variables are the angle of the pendulum θ and angular velocity $\dot{\theta}$. The way the observed system state $\mathbf{y} = [\theta, \dot{\theta}]^T$ evolves over time can thus be generally described as the following differential equation:

$$\frac{d\mathbf{y}(\mathbf{t})}{dt} = f(\mathbf{y}(\mathbf{t}), t). \tag{3.1}$$

Note that the above equation is written generally and includes a time dependency, not only of the system state but also of the evolution rule $f(\cdot)$ itself. However, for some systems the evolution rule is independent of an external unknown input, in which case we refer to *autonomous dynamical systems*. Whenever the evolution rule depends on an external and undefined input, and thus implicitly time, the system is called a *non-autonomous dynamical system*. The set of all possible combinations of state variables (in our example θ,$\dot{\theta}$ and t) that affect the dynamical system is called the *state/phase space*. It describes a *manifold* in which one can move from one state to another.

3.1.1 Attractor

Most physical DSs in the real world tend to dissipate energy to their surroundings, for instance by friction and thermodynamic processes. Even when an external force is introduced, the system's energy loss combines with this external force to suppress the initial transients and results in a 'normal' behavior. In other words, most real world systems have damped dynamics. This 'normal' behavior corresponds to a sub-set of the state space, and the part on the manifold to which the dynamics are drawn is called the *attractor*. In the non-autonomous case the manifold can be different from one time step to another. In that case, the notions described in this section still hold when they are accompanied by a specific time constraint. In contrast, it is possible that a sub-set of the state space exists from which points within its neighborhood move away. Such a sub-set behaves like the opposite of an attractor and is called a *repeller*.

Basin of attraction

The basin of attraction is a neighborhood around the attractor consisting of all the states that eventually converge to this attractor.

Types of attractors

Attractors can have many shapes in the state space of the DS. However, in this section a brief overview of the most common attractor types is provided.

fixed point attractor: A fixed point attractor is a single point (and thus a single state) in the manifold of the state space in which the DS remains. This means that the evolution rule $f(\cdot)$ in Equation (3.1) is equal to zero resulting in an unchanged state. All states within its basin of attraction will eventually converge to this single state. If one considers a damped pendulum, the fixed point attractor corresponds to the downwards rest position, which is a stable equilibrium position. The manifold of this pendulum also contains a non attracting fixed point, which corresponds to the upwards and unstable equilibrium position.

limit cycle attractor: A limit cycle attractor is a periodic (closed) trajectory in the manifold to which points from both sides of the trajectory converge. An example of a system with this behavior is a pendulum clock. Given the random initial state of the pendulum, the amplitude and speed will eventually converge to a precise periodic movement that depends on the length and mass of the pendulum itself. In contrast, an ideal pendulum (no damping) has no limit cycle attractor because given the initial state the pendulum will move on a closed trajectory that will include this initial state. As every

initial state has its own closed trajectory they are not attracted to an isolated attractor.

chaotic attractor: Sometimes the DS can behave unpredictably[1]. Such chaotic behavior can even be present in very simple systems (e.g., logistic map). Although they seem to be a random process, they are fundamentally determined by the dynamical system's evolution rule (Provenzale et al., 1992).

Bifurcation

If the evolution rule in Equation (3.1) depends on a certain parameter, the manifold of the corresponding DS will also depend on this parameter. Normally, changing the value of this parameter slightly causes no dramatic changes to its state space. However, in some cases, a small change in parameter value suffices to cause a qualitative change to the structure of this state space. Such a transformation is called a bifurcation.

3.2 Exploiting the dynamical system

In our goal of learning rich motion and motor skills from biological systems, we can parameterize the recorded motions by identifying them with a DS allowing for the generated motions to become adaptable. For instance, if we want to learn locomotion, which is often a *periodic* movement, we can adapt a DS (such as Equation (3.1)) in such a way that a limit cycle attractor exists with the desired shape, amplitude and frequency. Reaching movements, on the other hand, are called *discrete* because at some point in time the movement stops. Such stopping behavior can be generated by a DS with a fixed point attractor. However, the shape of the manifold needs to be modified in such a way that before reaching the target, the movement resembles the recorded motion.

Dynamic Movement Primitives

One of the most popular approaches that uses the concept of learning a dynamical system to expand a limited set of recorded motions to a full parameterized range of motions is called Dynamic Movement Primitives (DMPs). DMPs were introduced by Schaal et al. (2000). Here, the attractor landscape was shaped so that its basin of attraction had certain desired properties. The key contribution of this approach is the formalization of nonlinear dynamic equations so that they can be flexibly

[1] Not all unpredictable behavior can be considered as being chaotic. For example, a quasiperiodic attractor is unpredictable but not the cause of chaotic behavior.

adjusted to represent arbitrarily complex motor behaviors, without losing the stability of the system. Here, a globally stable linear DS with a unique fixed point attractor is used that converges exponentially towards this fixed point. By augmenting this system with a nonlinear function $h(\cdot)$, this exponential trajectory can be converted into a more complex and desired trajectory. To guarantee monotonic global convergence, an additional canonical DS (linear) is used to find the appropriate $h(\cdot)$. In Schaal et al. (2000), it has been shown that the combined system converges asymptotically to the unique point attractor. Learning periodic motions, such as in Ijspeert et al. (2002) is also possible with DMPs. However, to the best of my knowledge, combined learning of both a limit cycle and a fixed point attractor within a single system has not been demonstrated. Furthermore, for each degree of freedom (DOF), a separate DS needs to be learned, which means that learning implicit coupling between two DOF becomes impossible. Defining an explicit coupling between two separate motions, on the other hand, is possible. DMPs and variants are extensively used to learn both discrete and periodic motions.

Stable Estimator of Dynamical Systems

Another approach, the Stable Estimator of Dynamical Systems (SEDS) was proposed by Khansari-Zadeh and Billard (2011) to model a motion as a nonlinear autonomous DS under sufficient conditions to ensure global asymptotic stability at the target. Similar to DMPs, the DS is exploited so that all generated motions follow the demonstrations closely. With SEDS it becomes possible to encode several discrete motions into a single globally stable dynamical system. However, with SEDS a static mapping is created that prevents it to learn motions with intersecting parts. At such an intersection, SEDS is unable to distinguish between both possible directions. It has been demonstrated that using velocity information as an additional input overcomes this problem. However, the learning of periodic motions has not been demonstrated.

3.3 Reservoir Computing approach

I have mentioned before that we can consider an ESN to be a dynamical system. To exploit the network dynamics in a similar manner to DMPs and SEDS, we need to train the network so that the desired attractive behavior is achieved. The topology described in the previous chapter is used (shown in Figure 2.1) to learn a DS that maps the current position, velocity and/or acceleration value onto the next one. I will first consider the autonomous case where no external input is used ($\mathbf{W}_{\text{i}}^{\text{r}} = \mathbf{0}$) except for the output feedback. To show that using output feedback and no input

corresponds to an autonomous DS, we can write Equation (2.3) with $\gamma = 1$ as:

$$\mathbf{a}(k+1) = \tanh\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{a}(k) + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{o}(k) + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}\right) \tag{3.2}$$

$$= \tanh\left((\mathbf{W}_{\mathrm{r}}^{\mathrm{r}} + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{W}_{\mathrm{r}}^{\mathrm{o}T})\mathbf{a}(k) + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}\right). \tag{3.3}$$

The output feedback can thus be internalized into the reservoir itself by changing the reservoir weights ($\mathbf{W}_{\mathrm{r}}^{\mathrm{r}} = \mathbf{W}_{\mathrm{r}}^{\mathrm{r}} + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{W}_{\mathrm{r}}^{\mathrm{o}T}$), which demonstrates the autonomy of this network topology. Introducing leaky integrator neurons does not change this. In Sussillo and Abbott (2012), such internalization of the output feedback was investigated while retaining sparseness conditions for the reservoir weights. They argued that it is more biologically plausible from a neuroscience point of view that a neural network can sense its own output, instead of needing external output feedback.

To actually train the network, we need to decide which mapping we want to learn. It is possible to learn one or more of the following mappings $f(\cdot)$ with a single ESN where $\mathbf{o}(t)$ represents the recorded position:

$$\text{position:} \quad \mathbf{o}(t+dt) = f(\mathbf{o}(t)) \tag{3.4}$$

$$\text{velocity:} \quad \dot{\mathbf{o}}(t) = f(\mathbf{o}(t)) \tag{3.5}$$

$$\text{acceleration:} \quad \ddot{\mathbf{o}}(t) = f(\mathbf{o}(t)). \tag{3.6}$$

Each mapping has its own advantages and disadvantages. For instance, position mapping only controls the position and the velocity of the resulting motion can fluctuate. In contrast, when learning velocity mapping, similar fluctuations can cause the resulting movement to fail to reach the target position. When controlling robots, the velocity, position and even the acceleration should be tightly controlled. Therefore, it is advisable to learn multiple mappings in order to produce a smooth and accurate control signal. In the case of the velocity and acceleration mapping, the generated output should be integrated before it can be used as a feedback signal. As this complicates the explanation, I will limit this chapter to a description of the position mapping.

Usually task space movements recorded from human motions are in 6 dimensions (3 dimensions for position and 3 for rotation). However, for visualization and interpretation purposes the description in this chapter is limited to 2 output dimensions that describe the position in a plane. To train the network to produce the next (x,y) position given the previous one, the recorded data needs to be provided as the target output. As previously explained, when using ridge regression the feedback line is clamped with the same training data as the output, unlike FORCE learning where the actually produced output is fed back. Both learning algorithms described in Section 2.1.5 can be applied to train the output weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$ of the network; however, the description and evaluation of the proposed MPG will only be presented

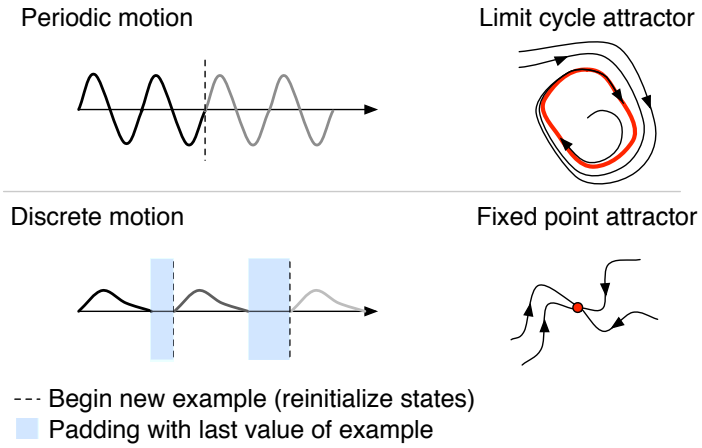


Figure 3.1: Illustration of the composition of the training data before it is used for training. Different examples of the same motion are shown for both periodic and discrete patterns. A dashed line indicates the moment at which the reservoir states are reinitialized. For discrete patterns, the gray areas before the dashed lines illustrate the added padding that consists of the last value of the associated discrete pattern. The corresponding attractor types are illustrated on the right.

for the ridge regression case. When using ridge regression, the feedback training data is shifted one time step compared to the training data used for the output. It turns out that the structuring of this recorded data and the actual neurons' states during training are important for achieving the desired attractor. In the following section we will describe this in more detail.

3.3.1 Training procedure

The training of a periodic pattern in the context of a CPG was investigated extensively by wyffels and Schrauwen (2009). Here, a periodic pattern was used for training. In the presented approach the training of periodic patterns is achieved in a similar manner. However, to allow for the training of different examples (generated by back-driving[2] the robot) the reservoir states are reinitialized after each example[3] ($\mathbf{a}(k_e) = \mathbf{0}$ with k_e the first time step of a new example). This reinitialization

[2] A back-drivable mode allows the robot to be moved by hand. As a result, the imposed movement can be captured by recording the encoder values.

[3] Containing multiple periods.

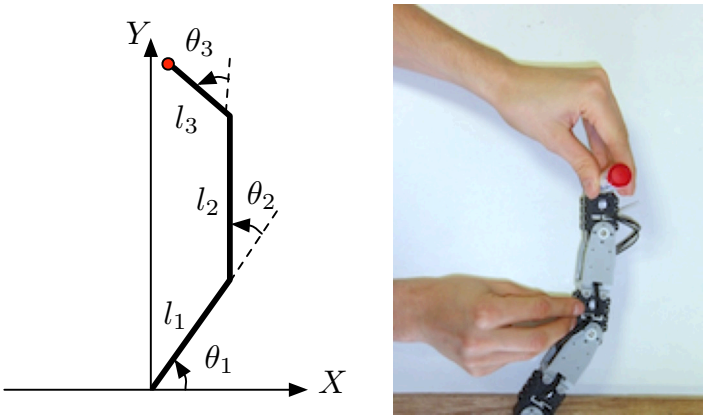


Figure 3.2: Plot on the left shows a schematic representation of the 3 DOF planar manipulator together with the associated angles θ_i and link lengths l_i, $i = 1,2,3$. The actual implementation of the robotic manipulator is shown on the right.

is denoted in Figure 3.1 by a vertical dashed line.

The training of a discrete motion differs in the sense that after each example a random sized padding of the last value of an example is added. By applying this padding, the output weights of the ESN are also trained on the transients of the demonstrated motion to its fixed target position. Similar to periodic motions, the reservoir states are reinitialized at the beginning of an example after the padding of the previous example. This process is illustrated at the bottom of Figure 3.1.

3.3.2 Robotic task

In order to evaluate the proposed motion pattern generator (MPG), I have conducted some experiments on a planar manipulator with 3 rotational joints. I used 3 joints to increase the complexity of the kinematics, although this was not the main focus of the experiments. As shown in Figure 3.2, the manipulator was constructed with Bioloid components and Dynamixel AX-12 servos that are controlled by a RS-485 serial bus. Several demonstrations of multiple motions were recorded by moving each servo by hand (in a back-drivable modus) and by reading the corresponding encoder values. These angular values were converted into a task space trajectory by applying a transformation from joint to task space called forward kinematics. The several resulting trajectories were saved and used as training data for the ESN. After training, the ESN's attractor landscape is shaped by the demonstrated task space trajectories and should be able to generalize beyond this training data. To al-

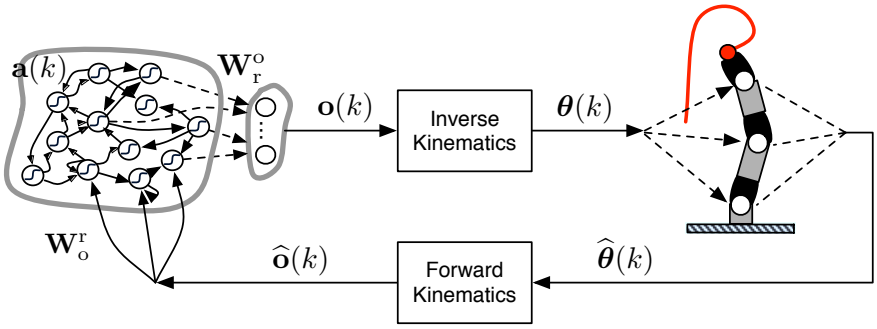


Figure 3.3: Illustration of the control loop in which the manipulator is controlled by joint angles $\boldsymbol{\theta}(k)$ that correspond to a task space position $\mathbf{o}(k)$. The hat symbol is used to indicate a measured value. The other symbols are consistent with those introduced in Figure 2.1.

low investigation of this property the robot is integrated in the feedback loop of the ESN allowing for a closed loop control. As shown in Figure 3.3, the control loop consists of the initial task space position of the manipulator, which can be measured by applying forward kinematics for the joint encoder values. This task space position is given to the trained ESN via its feedback connection, after which the ESN produces a new position. This generated task space position is converted to angular values by applying inverse kinematics. By using a P-controller these angular positions are converted to their corresponding current values that will actually move the robot's *end-effector*[4] towards its desired task space position. The necessary details for calculating the inverse/forward kinematics are presented in Appendix A.1.

3.3.3 Task and network settings

To train the ESN, several writing motions of the letters A,I,R,O,B,T and S were recorded. As mentioned above, each writing motion was demonstrated by moving the manipulator by hand in a back-drivable mode. During training, each letter was shown 7 times, each with different starting positions. In order to have the same target point for every example, each writing motion's example was subtracted from its last value. By doing so the target point of each example becomes zero. This makes the motion generation invariant of the desired target. In order to limit the magnitude of the values used for training the ESN, all training data is normalized as described in previous chapter. When applying the trajectory to the robot, the

[4]The part of the robot that interacts with the environment. This corresponds to the gripper when using a robotic arm.

Table 3.1: Network parameters

Parameter	Value
N	400 neurons
ρ	0.99
γ	0.3
f_b^r	0
f_o^r	1

ESN's output must be converted back to the correct range.

To demonstrate the ability to learn periodic motions, I have used 7 examples of a figure-8, not by back-driving the robot but by imposing the following equation:

$$\begin{aligned} x &= 13 + 3\cos(\psi) \\ y &= -5 + 3\sin(2\psi) \\ \psi &= t + \frac{\pi}{2}. \end{aligned} \tag{3.7}$$

Because every example of the figure-8 has the same equation, the need for state reinitialization can be omitted. Essentially, one long periodic pattern can be used for training.

As an alternative to training a separate ESN for each motion, it is possible to learn multiple motions in a single ESN. Learning multiple periodic patterns in a single network has been shown before in wyffels and Schrauwen (2009). In contrast, for discrete motions it is possible to shape the attractor landscape in such a way that they all converge to the same fixed point attractor. The initial position, however, determines which motion is generated before reaching this fixed point. The training of such a system is achieved by interleaving the examples of the different motions into a single dataset (Section 3.3.1).

For both motion types (periodic and discrete) the same network settings are used. The parameter values are shown in Table 3.1. In this ESN configuration, the optimal regularization parameter ν was determined by optimizing it with 4-fold cross-validation (Section 2.1.5.3). The other described parameters could be optimized by applying a grid search; however, the hand tuned parameters were sufficient for demonstrating the ESN's capabilities.

3.3.4 Generating Motion Patterns

For each letter, for which several demonstrations are available, a single ESN is trained.

After training, the generated motion is applied to the robot manipulator by converting task space positions to joint space positions according to the inverse kinematic model. Figure 3.4 shows the resulting trajectories, after training, for the I, R and figure-8 writing motions (shown in black). The shown trajectories are those that are generated by the robot manipulator by applying forward kinematics to the actual actuator positions in joint space. Due to small deviations in the precision of the internal PID controller, these produced trajectories are not smooth. However, the motions generated by the RC-networks are smooth.

In order to qualitatively evaluate the generalization capabilities, the generated motions for different starting positions are shown in Figure 3.5. The blue colored grid for each letter indicates the different starting positions from which a motion trajectory has been generated towards the fixed point attractor at $(0,0)$. These trajectories are colored from gray to black as time progresses to denote the generation direction and the actual resulting attractor. As a result, the final limit cycle attractor of the figure-8 motion is clearly distinguishable from the initial parts of the trajectory. One example of the demonstrated motions is shown in red for each letter as a reference.

To demonstrate the need for memory in the generation of a discrete motion, examples of the letters A and R that contain intersecting points are used. At such an intersecting point the direction of the motion generation (knowledge about previous positions) is necessary to allow successful completion of the motion. This means that the transients of the ESN, due to its recurrent connections and its leak rate, are critical for producing such writing motions. A periodic motion like the figure-8 clearly requires a certain amount of memory for the same reasons. Generating such motions with simple memoryless mapping from one position to the next would be impossible because there are multiple successors to the current intersecting position. SEDS (Khansari-Zadeh and Billard, 2011) uses memoryless mapping (Gaussian Process Regression), which makes it impossible to generate a limit cycle attractor. However, by adding second order information, such as velocity and acceleration, SEDS has demonstrated its ability to overcome the problem of self intersecting trajectories.

3.3.5 Robustness

To investigate the robustness of a trained ESN, the output feedback was perturbed for 10 time steps by holding the y-coordinate at a fixed position and moving the x-coordinate to another position. This is demonstrated in Figure 3.6 for a discrete motion, the letter R and a periodic figure-8 motion. The shown motion patterns are those that are generated by the ESN, not the ones produced by the manipulator. However, apart from producing less smooth trajectories, there is no difference with the unperturbed case. The top of the figure shows the generated motion in a 2D

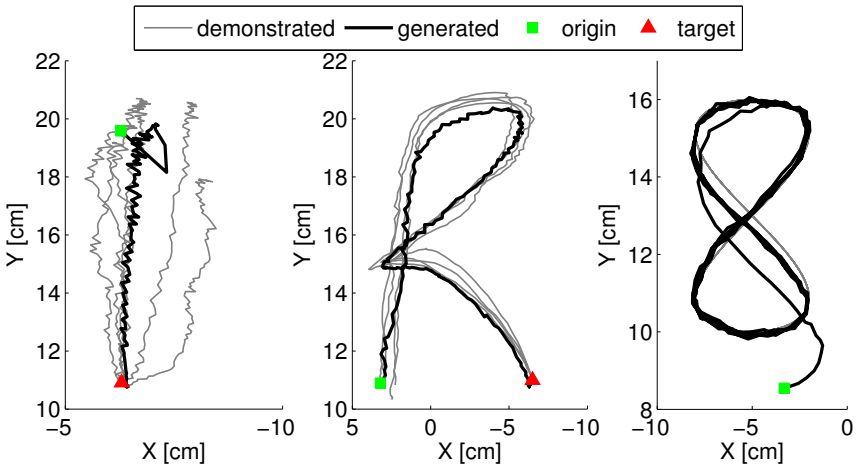


Figure 3.4: These plots show the demonstrated motions recorded by back-driving the manipulator in its compliant modus (gray lines). Furthermore, the actual produced manipulator motions after training on the examples are shown by black lines. The origin and target point of each motion is represented by a square and triangle, respectively. Because of the lack of a target position for periodic motions, only the starting point of figure-8 is shown.

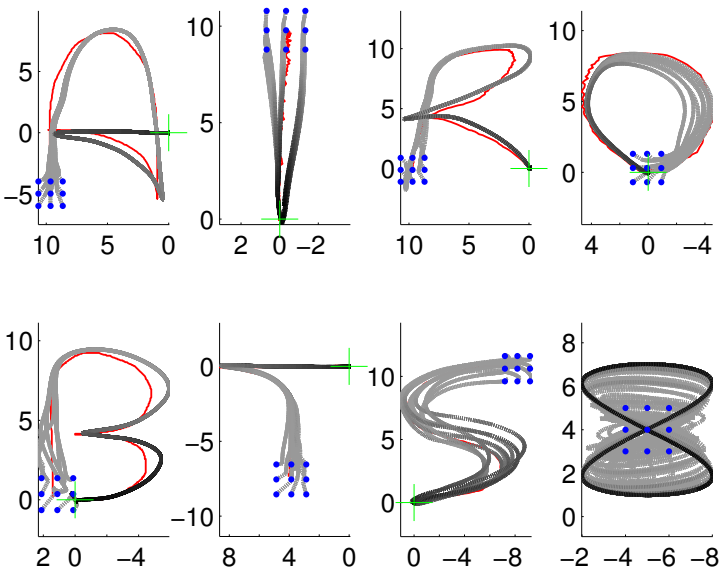


Figure 3.5: This plot shows the generalized motion trajectories for the ESNs trained with discrete patterns A,I,R,O,B,T and S, starting from 9 different locations (indicated by the blue dots) towards the same fixed point attractor (green cross). These trajectories are colored from gray to black as time progresses to give a sense of direction. This also allows one to distinguish more clearly between the final attractor behavior and initial parts of the generated trajectory. For the periodic figure-8 motion, only the starting positions are shown as its target is a limit cycle attractor. Additionally, a single example motion for each movement is shown in red as a reference. In some cases, however, this example is covered by the generated motions. The vertical and horizontal axis are both in cm and represent the Y and X-axis, respectively. I subtract the last value from each writing motion so that the final target is located at $(0,0)$.

plane. To reduce the size of the plot the horizontal axis represents the Y-axis. The other plots illustrate the effects of a perturbation on each dimension separately.

For the discrete motion, two perturbations were introduced, for two generation attempts respectively (one illustrated by a gray line, the other by a black one). The perturbation point at which each motion was held during 10 time steps is illustrated with a triangle (the perturbation is at time step 30 for the gray pattern and 80 for the black pattern). These described plots demonstrate that the motion generation is robust against perturbations and that after a certain amount of transient behavior the target position is reached. Furthermore, it is evident that the motion remains in its fixed point attractor for as long as the experiments last (300 time steps).

For the evaluation of periodic motion generation, the motion was again perturbed at different time steps (100, 338, 516 and 690). The perturbation frequency was small enough to allow convergence to the limit cycle attractor between two perturbations. To illustrate the transient motions, a different color was used between every two perturbations. It is evident that, after some transient behavior, the motion converges back to its limit cycle attractor. Additionally, because of this limit cycle attractor, the periodic motion will continue until the experiment stops (600 time steps).

Using a dynamical system to represent a recorded motion, allows for the motion generation to be adapted based on feed back of the actual manipulator. Thanks to the closed loop control of the robotic manipulator, the motion generation is put on hold when the end-effector is interrupted or pulled towards a resting position. Although a new target motion position is given to the interrupted manipulator, the feedback position does not change. The new desired position that is generated will thus depend on this feedback information, causing lower control gains than when a motion is blindly commanded to the robot (open loop). Due to these lower control gains, interacting with the manipulator will become less dangerous. Holding the manipulator in a fixed position can disturb the memory of the ESN. However, this anticipated effect was not visible in the conducted experiments. Depending on the length of the ESN's memory, holding the robot manipulator fixed exactly at an intersection point might cause it to loose its sense of direction.

3.3.6 Modulation

So far I have demonstrated that an ESN is able to shape its basin of attraction according to recorded demonstrations, and as a result it can generalize beyond them. The general dynamic behavior of the DS is of course not only regulated by the training of the output weights; adding an input to the network, regulating the leak rate or controlling the bias of each neuron independently influences the properties of the motion pattern generator as well. When these modulation signals change over time (i.e., they are not constant), the resulting basin of attraction is in fact affected

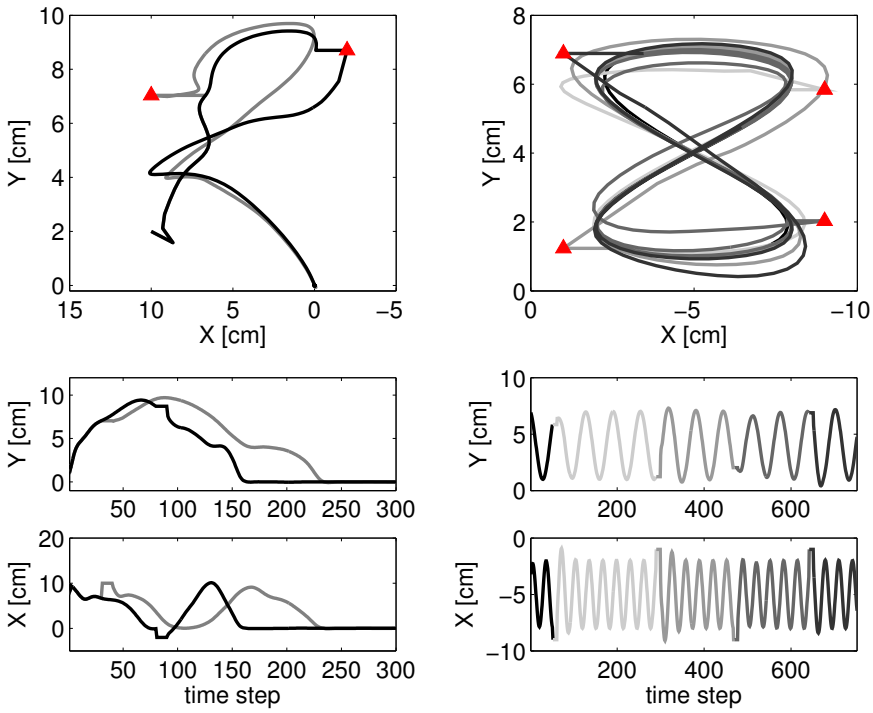


Figure 3.6: These plots demonstrate the effect of perturbations on the generation of both discrete (left) and periodic patterns (right). The top plot illustrates the robustness against such perturbations in a 2D plane. The plots at the bottom illustrate the robustness for each dimension. A perturbation is introduced during 10 time steps by keeping the y-coordinate constant but dragging the x-coordinate to a certain position (illustrated by the triangles). For the discrete pattern, 2 generation attempts for the letter R are shown (in black and gray) with a different perturbation. For the periodic motion, 4 perturbations were applied at the same motion. Between each perturbation (time steps 100, 338, 516 and 690) the generator has time to converge back to its limit cycle attractor. After each perturbation the color is changed to a different shade of gray.

by a unknown input signal and therefore the dynamical system is non-autonomous. In the following section I will give a brief overview of the modulation effects upon the attractor landscape.

3.3.6.1 Input driven modulation

Adding an input $\mathbf{u}[k]$ to the network allows the ESN to learn the relationship between a certain input and a desired behavior at the output of the pattern generator. For instance, if we want to switch from one periodic motion to another we can assign a different input value to each motion. To illustrate this, the presented MPG has been extended with a one dimensional input so that a discrete motion is generated for one input value and a periodic pattern for another.

The training data for this task was constructed by interleaving periodic and discrete motion examples, with padding and state initialization included, and setting the input to 0 or 1 for a discrete or periodic example, respectively. All training parameters were kept the same as in Table 3.1. However, because of the added input a hand-tuned input scaling factor f_i^r of 0.7 was used.

The top of Figure 3.7 presents the results of the experiment. The plots at the bottom show each dimension separately, together with the timing of the input switching (dashed line, low when $\mathbf{u}[k] = 0$ and high when $\mathbf{u}[k] = 1$). The arbitrarily chosen initial position in this experiment is located at $(0, 20)$ and $\mathbf{u}[k] = 0$. After some transient behavior (light gray color), the generation of a self intersecting letter R is achieved during the convergence to its fixed point attractor. This trained writing motion is shown in black. After switching the input (from 0 to 1) and some transient behavior, the ESN goes to its limit cycle attractor until the input is switched again to 0. Interestingly, due to symmetry[5] in the network, the generated discrete pattern is a rotated version of the trained one. In Figure 3.8 the generalization for a large grid of initial positions is shown. Again the intensity of the trajectory increases as time progresses. Panels A and C show the attractor landscape of the ESN with a 0 and 1 input, respectively. In contrast, Panels B and D give an indication of the attractor landscape of an ESN with added bias ($f_b^r = 0.2$). It is clear that the symmetry in panel A is removed by the added bias.

The velocity of the discrete and periodic motions depends on the velocity of the training examples; however, the transients between these patterns are unpredictable and can therefore exhibit large velocity bumps. Learning the velocity profile of the desired motion with the same ESN will reduce such undesired behavior.

It is clear that the change in input value causes the attractor landscape of the trained ESN to go through a bifurcation. Often a bifurcation plot is used to get an indication of how the attractor landscape is transformed in function of a parameter.

[5] When no input bias is used, a symmetry in the network states exists when the input is shifted in sign.

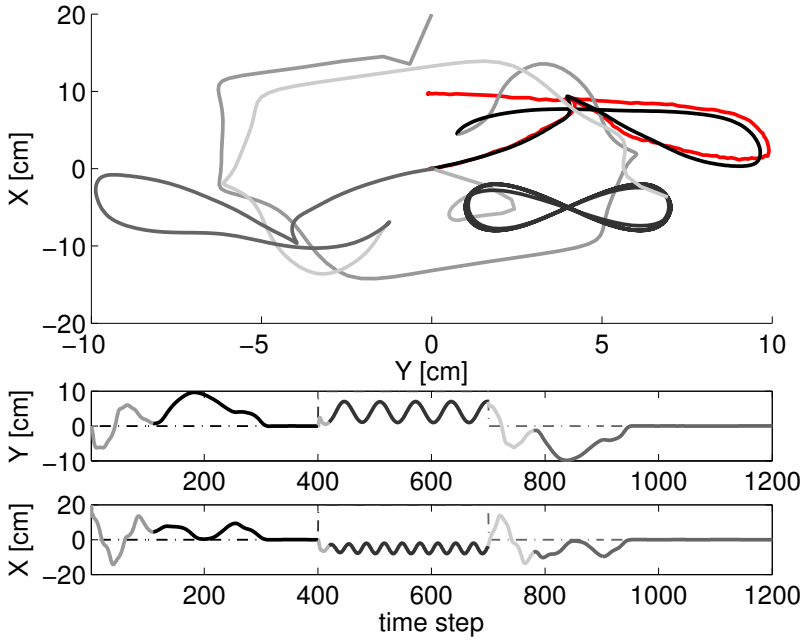


Figure 3.7: These plots demonstrate the switching capabilities between discrete and periodic patterns. The dashed line in the bottom two plots illustrates the switching moment, set by changing the network input (time steps 400 and 700), between the generation of the letter R a figure-8 and again a letter R. The transients between these patterns are shown in a lighter gray color while the discrete and periodic patterns are shown in darker gray. The red curve represents an example of the letter R.

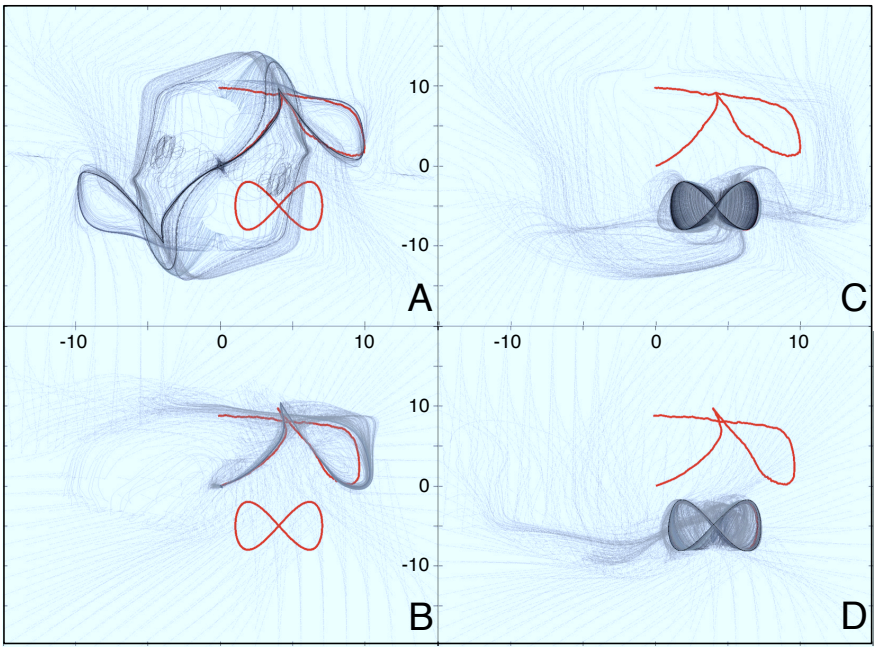


Figure 3.8: The top panels show the attractor landscapes for an ESN without bias $f_b^r = 0$ for an input set to 0 (panel A) or 1 (panel C). The bottom panels (B and D) demonstrate an ESN with bias $f_b^r = 0.2$. An example motion of both the letter R and figure-8 is shown in red as a reference. All axis units are in cm.

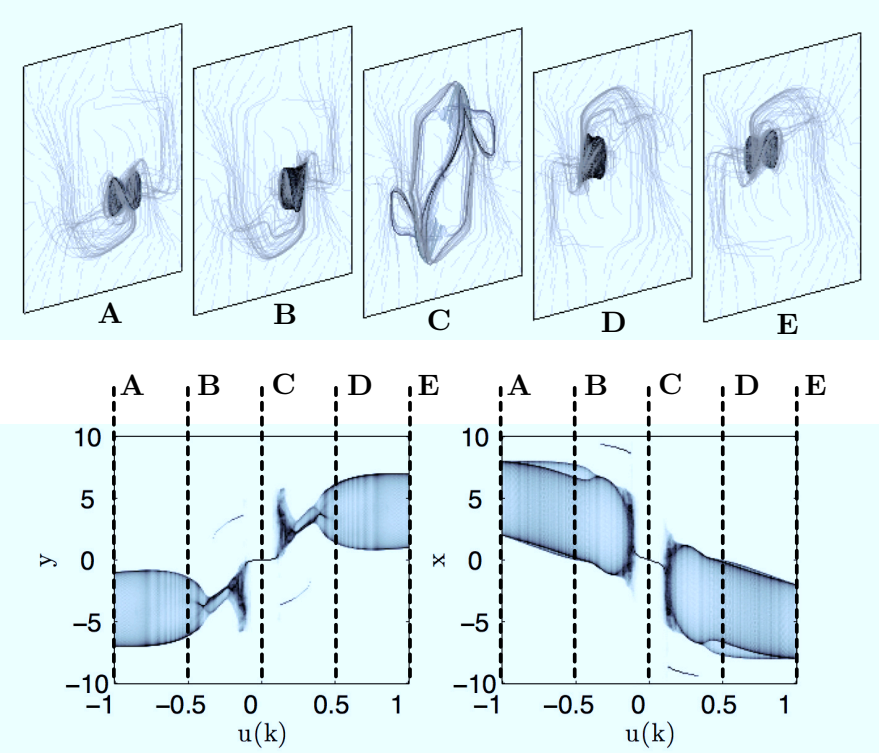
In order to create such a plot, the trained ESN for a given input was initiated from 400 different positions. For every starting position, the ESN was allowed to generate a trajectory of 1000 samples. Next, for every trajectory, the average value of the last 500 samples was taken. Finally, a histogram was created from these values for that particular input value. The actual values of this histogram are denoted by the gray scale in the plot. Figure 3.9(a) shows the bifurcation plot of the symmetric ESN (without bias) for each motion dimension. If the input is around -1 the attractor landscape contains a single limit cycle attractor. When the input gradually increases towards 0 this limit cycle slowly disappears until, after several bifurcations, a single fixed point attractor is reached at the 0 input. When the input value is increased even further, the symmetric limit cycle reappears after some bifurcations. For a few input values, the corresponding attractor landscape is shown at the top. In Figure 3.9(b) a similar bifurcation plot is shown, but now for an ESN with bias. In the left bifurcation plot there is a single fixed point attractor when the input is set to -1.

When gradually increasing the input value towards 0 several bifurcations occur, including one where two different fixed point attractors are present. When reaching 0, the attractor landscape only contains the fixed point attractor to which the desired discrete motion converges. Increasing the input further will cause the ESN to experience bifurcations again, until a limit cycle attractor is reached that corresponds to the periodic motion behavior. The motion generation is robust against input noise, as long as this noise is not causing the occurrence of a bifurcation.

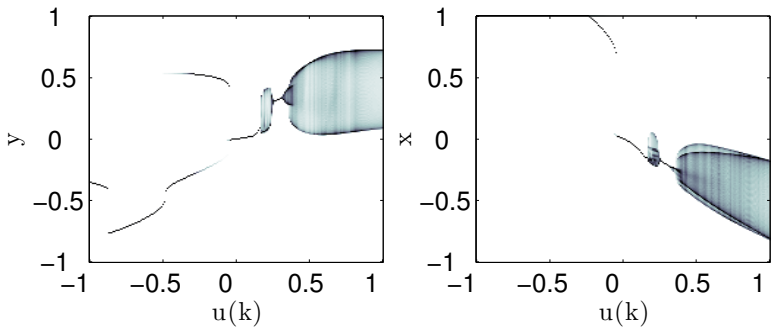
3.3.6.2 Changing bias weights

In Li and Jaeger (2011), a new method for modulating the shape of a learned periodical pattern was illustrated based on tuning the bias weights of the neurons instead of using additional inputs. In summary, the influence of adding a small bias to each neuron is determined after training. Therefore, each neuron is perturbed separately with a small constant bias. After perturbing each neuron, one can observe the influence of this on the properties of the output signal. For each property that one wants to modulate (e.g., amplitude, phase, shape) a control vector can be composed that can be used to modulate the output signal. Frequency modulation of the periodic motion, on the other hand, is more difficult to achieve. In more recent work by my colleague (wyffels et al., 2013), we proposed a generic method for modulating the frequency. This solution is based on the insight that there is a mutual dependency between the frequency of a periodic pattern and the geometrical shape of the attractor's orbit[6]. As a result, the orbit's shape can be changed to modulate the frequency. In Chapter 5 this method will be used to modulate an MPG that controls a robot

[6]Orbit: the curved and closed path within the basin of attraction that forms the limit cycle attractor.



(a) Without input bias: $f_b^r = 0$



(b) With input bias: $f_b^r = 0.2$

Figure 3.9: This Figure shows the bifurcation plots of two trained and input driven ESNs for each dimension of the motion (X,Y). The ESN in panel (b) uses input bias. The ESN in panel (a), however, does not use input bias and as a results exhibits symmetric behavior. In order to obtain a better understanding of how these bifurcation plots are created, I added a visualization of the corresponding attractor landscape for a few input values. The corresponding position in the bifurcation plot is indicated by vertical dashed lines. The vertical axis units are in cm.

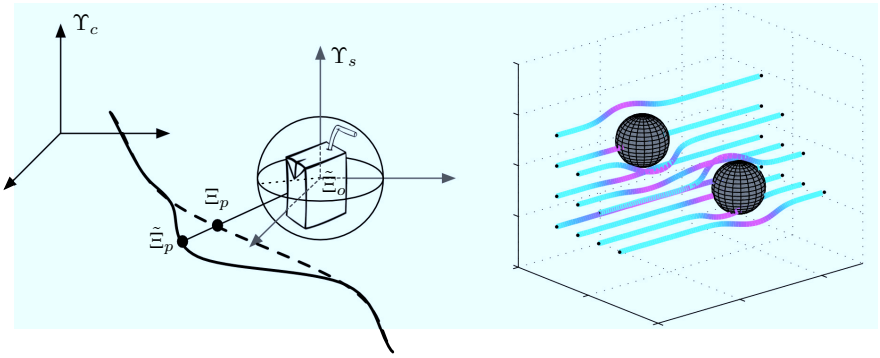


Figure 3.10: The left figure illustrates how a generated motion path is modified to avoid an object. The Cartesian and spherical frames of reference are indicated by Υ_c and Υ_s, respectively. The dashed black line illustrates the original path, while the modified path is illustrated by a solid line. The right plot demonstrates the obstacle avoiding behavior in 3 dimensions and in the presence of multiple spherical obstacles. The coloring of the lines changes from cyan to purple when the obstacle avoiding algorithm intervenes.

leg.

3.3.7 Obstacle avoidance

After training an MPG, the basin of attraction of the dynamical system is shaped according to its training examples. Due to the regularization used, this shaping has an effect on the entire dynamical system even outside the regions in which the training data is located. As a result the MPG can generalize beyond its training examples which makes anticipation for all possible situations unnecessary, especially when closed loop control is desired.

When a robot on which such an MPG is applied interacts with its environment, it is possible that these motions are perturbed by an obstacle. We do not want to anticipate for all possible obstacle locations and train the DS with corresponding obstacle avoiding motion examples. Therefore, in Ajallooeian et al. (2010) I proposed a dynamical system approach to obstacle avoidance, where the trained dynamics are transformed in the presence of an obstacle. More formally, we need to find the nonlinear function $\chi(\cdot)$ in the following equation with respect to Equation (3.5):

$$\mathbf{o}(t+\Delta t) = \chi(f(\mathbf{o}(t))), \tag{3.8}$$

in order to shape the dynamics to avoid an obstacle. Furthermore, as obstacles are often introduced in very short time periods, it is important that the MPG can react fast enough. Inspired by vector fields in the presence of repellers (e.g., force fields, fluid flows), the generated motion can be considered as a stream of particles. Due to the forces caused by an obstacle, the particles will deviate from their initial direction and avoid the obstacle. In that case, neither the initial path nor any history of the particle state is necessary for avoiding the obstacle.

Now consider such a particle to be defined within a Cartesian frame of reference Υ_c. The coordinates of a particle p within Υ_c are denoted by $\Xi_p = [x,y,z]^T$. In order to simplify the problem, the shape of the obstacle is assumed to be a sphere, large enough to encapsulate the real obstacle. This is illustrated in the left part of Figure 3.10. While a particle is moving in the direction of an obstacle, it can be described according to a spherical frame of reference Υ_s that is located at the center of the obstacle. If the center of the obstacle is denoted by Ξ_o, then the corresponding transformation from Υ_c to Υ_s can be written as:

$$[r,\theta,\varphi]^T = \Gamma(\Xi_p - \Xi_o). \tag{3.9}$$

Consequently, each particle within Υ_s can be described by a radial distance r, polar angle θ and azimuth angle φ. In this frame of reference it is much easier to condition the motion of the particles when they enter the vicinity of the obstacle (origin). Although many solutions are possible, I was inspired by the laminar flow of fluids around spherical obstacles. In the field of fluid dynamics, such flows are often explained by incompressible steady Stokes flows (Acheson, 1990), under the assumption that all surfaces are smooth and the particles do not interfere with each other. Furthermore, vorticity is omitted, otherwise the resulting motions would show vortexes after the obstacles. Such Stokes flows are regularly used in a wide variety of applications (e.g., determining electron charges, explaining the physics of aerosols). According to this model, where φ is assumed to be 0 at all times, the resulting velocity components are described by:

$$\frac{dr}{dt} = V\cos(\theta)\left[1 + \frac{R^3}{2r^3} - \frac{3R}{2r}\right], \tag{3.10}$$

$$\frac{d\theta}{dt} = -V\sin(\theta)\left[1 - \frac{R^3}{4r^3} - \frac{3R}{4r}\right], \tag{3.11}$$

with R the radius of the spherical obstacle and $V = \|\frac{d\Xi_p}{dt}\|$ (i.e., the norm of the particle's velocity) (Kundu and Cohen, 2008). Blindly applying this model will not result in the desired behavior because particles going precisely through the center of the sphere can get caught within the sphere, yielding a very large $\frac{dr}{dt}$. Instead of increasing $\frac{dr}{dt}$ the closer the particle gets, $\frac{dr}{dt}$ should converge to 0 when the particle

comes close to the obstacle. Therefore, the following expression for $\frac{dr}{dt}$ is proposed:

$$\frac{dr_+}{dt} = \begin{cases} 1-\tanh^2(-r_- + R + 1)\frac{dr_-}{dt}, & r \leq R+1, \\ 0, & \text{otherwise}, \end{cases} \tag{3.12}$$

where $r_.$ denotes before $-$ or $+$ after the changes. Furthermore, a similar expression for $\frac{d\varphi}{dt}$ is used as Equation (3.11):

$$\frac{d\varphi}{dt} = -V\sin(\varphi)\left[1 - \frac{R^3}{4r^3} - \frac{3R}{4r}\right], \tag{3.13}$$

because a generated motion can approach an obstacle from any direction, even those where $\varphi \neq 0$. After modifying $[r,\theta,\varphi]^T$ with $[\frac{dr}{dt}, \frac{d\theta}{dt}, \frac{d\varphi}{dt}]^T$, respectively, the updated particle position in Υ_s is converted back to $\tilde{\Xi}_p$. The nonlinear transformation $\chi(\cdot)$ in Equation (3.8) can thus be defined as:

$$\chi_i(\Xi_p) = \tilde{\Xi}_p = \Gamma^{-1}(\Gamma(\Xi_p - \Xi_{o,i}) + \Delta t[\frac{dr}{dt}, \frac{d\theta}{dt}, \frac{d\varphi}{dt}]^T) + \Xi_{o,i}, \tag{3.14}$$

for the i-th obstacle. The right plot of Figure 3.10 demonstrates the avoiding behavior in 3 dimensions for a set of stream lines. As demonstrated, it is possible to handle multiple objects by applying Equation (3.14) recursively (e.g., $\chi_2(\chi_1(\Xi_p))$). However, when two objects are too close to each other, it is advisable to treat them as one large obstacle, because the influence of one obstacle can overcompensate the changes made by another obstacle.

As mentioned previously, the geometry of the obstacle is simplified by a sphere around the object. However, for some objects, especially large flat objects, this simplification is imprecise because a lot of space around the object is incorrectly marked as an obstacle. An ellipsoid could be a good alternative, but for some objects this is also insufficient. Instead of defining a fixed R for an obstacle, however, it is possible to dynamically change the radius of the sphere in such a way that it fits the actual object better. To prevent local cavities in such a representation, a convex hull that is defined in spherical coordinates by a dynamic radius $R(\theta,\varphi)$ and a set of angles would be a good choice.

The obstacle avoiding strategy described above allows a motion position at a given time (a particle) to be updated if required, without the need to know the entire motion trajectory/plan from the beginning. This property allows for the approach to be applied in real time. Within the European AMARSi project this obstacle avoidance strategy was evaluated in (Khansari-Zadeh et al., 2010) and shown to be superior in all tasks.

One of our partners within the European AMARSi project has investigated the same ideas and applied them on several robotic tasks (Khansari-Zadeh and Billard, 2012). Here, SEDS was used to learn the stable generation of demonstrated

motions.

3.4 Conclusions

In this chapter I have presented the design of a motion pattern generator (MPG) from a dynamical system point of view. Based on the underlying concepts of DMPs as introduced by Schaal et al. (2000), I have investigated how the dynamics of an ESN can be exploited for the generation of both periodical and discrete movements. The attractor properties that make such generation possible have first been introduced. After describing the necessary training procedure, the MPG was applied on a planar robotic manipulator. Next, I have demonstrated the MPG's ability to successfully shape its attractor landscape according to the writing motion of the letters A, I, R, O, B, T, S and a periodic figure-8. Furthermore, the generalization towards different starting positions was demonstrated, as well as the robustness of the generated pattern in the face of perturbations. The MPG's modulation possibilities were also presented, and some results were shown.

As a consequence of applying the MPG to the manipulator in a closed loop, the motion generation is interrupted when the manipulator joints are obstructed[7]. The new position will thus depend on the feedback of the robot and as a result the control gains will be much lower than when controlling the robot in open loop. This property makes the use of a DS as a motion generator particularly attractive for applications where human robot interaction is required.

Finally, an obstacle avoidance strategy was discussed, which shapes the dynamics of the MPG in such a way that introduced obstacles are avoided. One of the main advantages of this obstacle avoidance strategy is that it reshapes the generated motion during the generation itself (i.e., real time), in a similar manner to air flowing around an airplane. This makes it especially useful in robotic tasks, where fast obstacle avoidance is desired.

Within the AMARSi consortium, the described obstacle avoidance approach was used together with an early version of the presented MPG. Its performance was evaluated on a set of benchmark tasks and compared to other approaches (Khansari-Zadeh et al., 2010). The results showed, that simultaneously controlling the velocity and position is important to prevent undesired velocity bumps. Furthermore, it was demonstrated that the described obstacle avoidance strategy was successful in all tasks.

[7] Even when an obstacle avoiding algorithm is incorporated, for instance in the task space motion, the joints can still be obstructed.

4
Adaptive Control Framework

The previous chapter described the design of a motion pattern generator (MPG) that shapes the basin of attraction of a dynamical system (i.e., an echo state network) in such a way that the recorded motions become a part of the attractor landscape. This manifold shaping also affects regions around the examples which enables the MPG to generalize beyond the recorded example trajectories. As demonstrated, this generalization ability allows the MPG to control a robot in closed loop, where the actual joint space positions (or task space position) measured by encoders can be used to provide the MPG feedback on the robot's state. Given this feedback the motion generation is adjusted to prevent high control gains due to large position errors. However, the control performance then depends heavily on the kinematic and/or dynamic model of the robot. For most commercial robots that one can buy, an accurate kinematic model is available. For compliant robots that are built with elastic materials, however, such kinematic model becomes inaccurate or at least difficult to acquire. More and more research is being devoted to dynamic control because often used PID controllers, which convert a desired angular position to a torque, yield high control gains and thus inherently dangerous behavior. Furthermore, if control gains can be kept low without having to trade in to control accuracy, the energy use of the robot will be much lower. Having an accurate dynamic model is, however, much more difficult even for the most commonly used robots that are commercially available. Especially for compliant robots, where the mass distribution of elastic materials can change, the lack of a good model poses a real problem. In this chapter an Inverse Modeling Adaptive (IMA) feedback controller is proposed that learns to control a physical system, often referred to as plant, online and without having a predefined model. In this case, the physical system can be considered as being a black box with a few inputs and outputs. Here, the inputs are the controlled parameters and the outputs the observable states of the system. As we already know, most physical systems can be represented by a dynamical system following a fixed evolution rule. The underlying idea of the IMA controller

is that another dynamical system, such as an ESN, can be used to learn an inverse model of the plant, mapping the observable state onto control commands. When the adaptive controller succeeds in learning an inverse model, it can be used to produce the corresponding control parameters in order to reach the target state. In this chapter, I will use an ESN as basic learning modules but the framework itself is general. Any machine learning technique that is able to model temporal functionals online could be used: tapped-delay line models with non-linear regression or neural networks, regular RNNs, Long Short Term Memory RNNs (Hochreiter and Schmidhuber, 1997), etc.

4.1 Design of the control framework

Every day we use machines that depend on control mechanisms in order to work properly. Two of the most widely applied control schemes are feedback controllers and feed-forward controllers. As illustrated in Figure 4.1(a), Feed-forward controllers use a predictive model to anticipate for the effect of measured disturbances and takes corrective actions in order to achieve the desired result. As shown in Figure 4.1(b), feedback controllers, such as the one proposed in this chapter, use measurements of the dynamical system (plant), compared with a desired value, to control the plant-input. For instance, the cruise control of a car uses a feedback controller to keep the velocity constant. When the car is driving downhill the car will go faster because of gravitation. The controller observes this increase in velocity and reduces the throttle to ultimately converge to the desired velocity.

Classical approaches of feedback controllers can be grouped by techniques that do or do not use prior knowledge of the plant. The latter, such as PID controllers, use no direct information of the plant dynamics. Other non-prior-knowledge-based techniques use a model exploration strategy where the observation produced by a random action is used by the controller to adjust its control. In the work by Jaeger (2002), such a strategy is taken. Here a Reservoir Computing (RC) network is used that is trained offline by using random values as training output and the plant response to these values as training input. In this example, the feedback information $\mathbf{y}(t)$ excites the RC-network in 2 versions: the current feedback $\mathbf{y}(t)$ and a delayed version $\mathbf{y}(t-\delta\Delta t)$. During training, the desired output (which is the random plant-input values $\mathbf{x}(t)$) are also delayed δ time steps before being used as training data of the RC-network.

The main reason for this delayed network input is to learn the necessary actions to bridge the gap between the present and future plant-output. After training the output weights (dashed lines in Figure 4.2(a)), the desired plant-output $\mathbf{y}_d(t+\delta\Delta t)$ is given to the input that was connected to $\mathbf{y}(t)$ during training. The actual plant-output on the other hand, is given to the input of the network that was connected

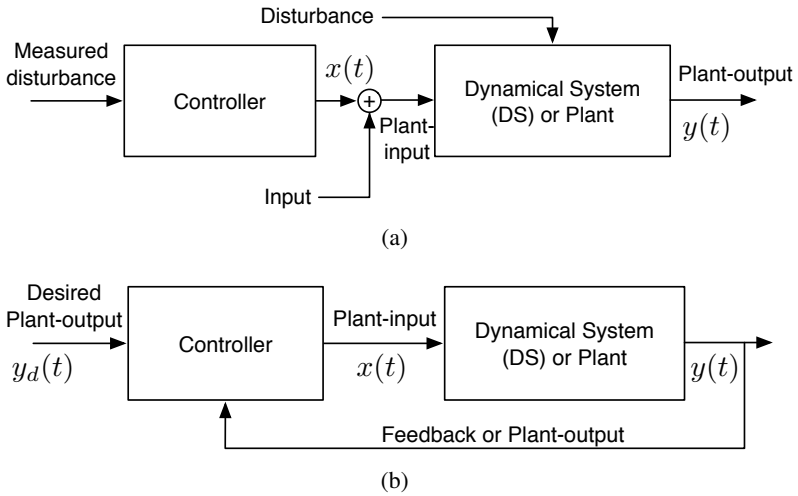


Figure 4.1: Illustration of (a) a feed-forward controller, (b) a simple feedback controller and a dynamical system (DS) or plant, accompanied by the used terminology. $\mathbf{y}(t)$ and $\mathbf{y}_d(t)$ represent the actual and desired plant output respectively. The output of the controller is denoted by $\mathbf{x}(t)$.

with $\mathbf{y}(t-\delta\Delta t)$ during training. As illustrated in Figure 4.2(b), the resulting network output $\mathbf{x}(t)$ drives the plant.

The idea here is to model the progress in plant-input $\mathbf{x}(t-\delta\Delta t)$ given the past and current plant-output ($\mathbf{y}(t-\delta\Delta t)$ and $\mathbf{y}(t)$). This model is used to determine the plant-input given the current and future plant-output ($\mathbf{y}(t)$ and $\mathbf{y}_d(t+\delta\Delta t)$) where the model is expected to generalize the trained behavior.

It should be noted that the choice of a good δ is essential to find such a model. δ and therefore the sample rate, determines the amount of time that the network has to reach the desired plant-output. However, in this work, it is assumed that the sample rate is predetermined and equal to one sample per integration time step. Therefore, the effect of δ depends on the dynamics of the plant. A smaller δ is used for a plant with fast dynamics and a larger one for a plant with slower dynamics. Finding a δ that corresponds to the plant dynamics, is essential in our control framework and is its main difficulty.

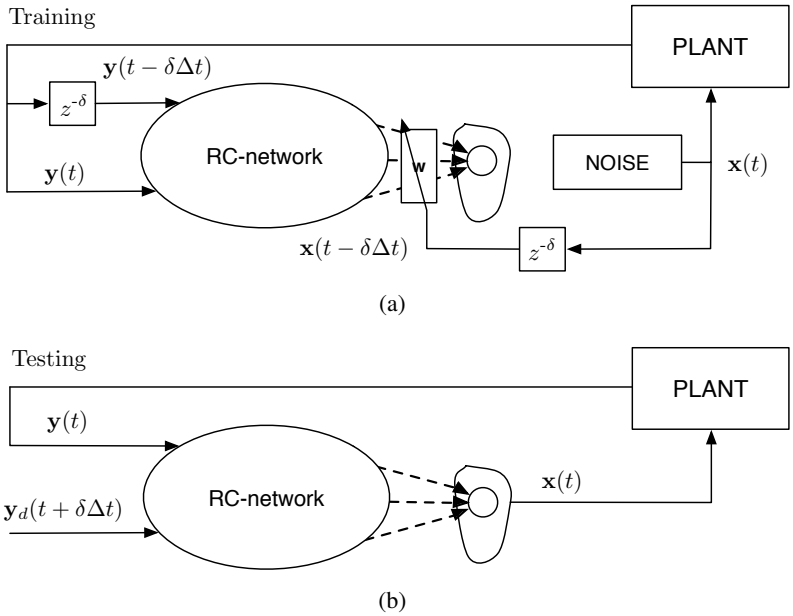


Figure 4.2: Illustration of a controller method described in the work of Jaeger (2002). During training, random $\mathbf{x}(t)$ values are used to train the output weights of the network based on the plant response $\mathbf{y}(t)$ on these values. Afterwards, during testing, the trained network is used to control the plant according to to the desired plant-output $\mathbf{y}_d(t+\delta\Delta t)$

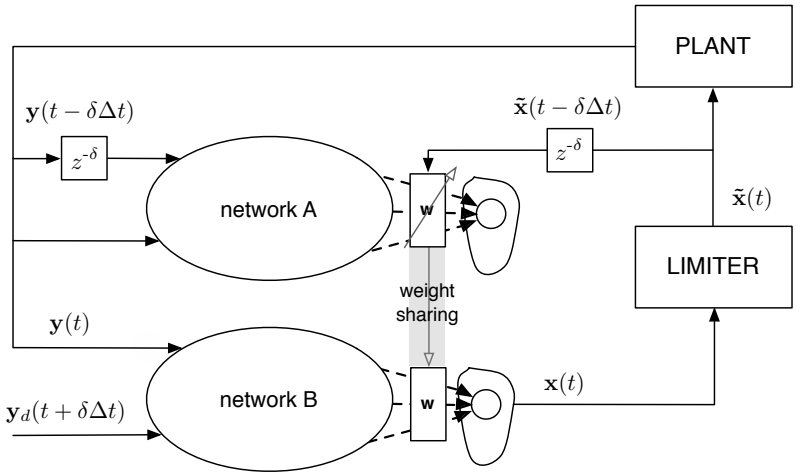


Figure 4.3: Schematic representation of the IMA controller. The dashed arrows represent the output weights $\mathbf{w}$ that are trained. These are the same for both networks (weight sharing). The optional limiter limits the values $x(t)$ to a desired range which, for example, represent imposed motor characteristics. Afterwards, the limited values $\tilde{\mathbf{x}}(t)$ excite the plant. The values $\tilde{\mathbf{x}}(t-\delta\Delta t)$ are used as desired network *A* output to train the weights $\mathbf{w}$. The resulting weights are used for network *B* as well.

4.1.1 Inverse modeling adaptive (IMA) control framework

In order to allow online control in which no prior knowledge (model) is necessary, the control framework shown in Figure 4.3 is proposed. This controller is called an Inverse Modeling Adaptive (IMA) controller, because of its adaptive nature and the fact that it creates an inverse model while interacting with the plant. Under the hood of this IMA controller, a similar network to the one described in Figure 4.2(a) is used. This network, called network *A*, is trained online in a supervised manner by using Recursive Least Squares (RLS). Next to network *A*, we have a duplicate network, network *B*, with the same input-network-and output weights (weight-sharing). This network is connected to the plant in a similar manner as the network in Figure 4.2(b). The output of this network is not only connected to the plant but is also used (delayed with δ time steps) as desired output $\mathbf{y}_d(t+\delta\Delta t)$ for the training of the output weights. The network states are initially the same for both networks and are randomly chosen according to a normal distribution ($\mathcal{N}(0,1)$). This random initialization is necessary to initiate the plant with random values. Without these values the amount of information necessary to train the internal model will be insufficient to generalize well. Because the inputs for both networks are not the same, the corresponding states will evolve differently. However, as network *A* is converging to a more accurate model, the inputs of both networks will converge to each other, but with a delay δ. Because of the desired plant output and the current plant feedback as input, network *B* starts generating values that drive the plant. For some plants it might be necessary to limit these values to a certain range. For instance, when controlling an actuator, the amount of torque that it can deliver is bounded. In Figure 4.3 this bounding is represented by a limiter that converts $\mathbf{x}(t)$ values to $\tilde{\mathbf{x}}(t)$. These values, delayed with δ time steps, are used as desired output of network *A*. With each iteration, the updated output weights are shared with network *B*.

By applying this topology, network *A* is learning a model solely on the observed plant-input and output during actual control. Network *B* on the contrary, uses the trained model to control the plant based on both the desired and actual plant response.

In Krose et al. (1990) a vision based neural controller is presented for a robot arm. Because of the fact that the desired and actual plant output are used as input, it shows some similarities with the IMA-controller. In work by van der Smagt (1995) this neural controller was extended to an online learning approach where the feed forward network weights are shared between a network that learns and one that controls. However, there the weight sharing was introduced as a necessity to handle the computational complexity of the training. For my ESN approach the used training algorithm is fast enough to update the weights online. In contrast, the

weight sharing of the IMA controller is motivated by the dynamics of the network state. When training an inverse model, it has an internal state that prevents it of being directly used to control the plant. This is the reason for introducing two parallel networks between which the trained output weights are shared.

As mentioned in Section 4.1.1, any dynamical system with a high dimensional state representation can be used in our discussed IMA control framework. However, to validate this framework on several tasks, a Reservoir Computing network, more specific an ESN (Chapter 2), will be used. Therefore, in the remainder of this chapter, network A and B refer both to an ESN. In general, such a network can at least be applied to all tasks that can be represented by a Volterra series if the pool of network states is rich enough (Maass, 2010; Boyd and Chua, 1985) (i.e. the network is large enough). In order to allow online learning of the output weights, the Recursive Least Square (RLS) algorithm, which was presented in Section 2.1.5.4, is used. As shown in Equation (2.16), RLS needs a target output $\mathbf{o}_{\text{target}}$ to train the network. As such data is unavailable at first, one can resort to the same "trick" described previously where random values for $\mathbf{o}_{\text{target}}$ and their corresponding plant responses are used as training data. Each iteration the updated output weights causes the model to improve resulting in a more accurate prediction of the control output.

Before the IMA control framework is evaluated on several control tasks, each with different properties, I will discuss the convergence and stability of the IMA controller.

4.2 Convergence and stability analysis

Controllers are made to interact with real life systems, from very complex systems such as robots to simple systems like the thermostat that regulates the room temperature. Unstable behavior of such controllers can therefore have large consequences. Whenever a new controller is proposed it is a generally accepted requirement to discuss the convergence and stability properties of this new controller. I start this section therefore by investigating the convergence of the RLS algorithm. Under the assumption that we have a one-dimensional target signal, the convergence of the error $e(k)$ in Equation (2.17) can be analyzed by rewriting this Equation as:

$$\mathbf{\Phi}(k)^T \mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k-1) = e(k) + \mathbf{o}_{\text{target}}. \tag{4.1}$$

As in Sussillo and Abbott (2009), we substitute this into Equation (2.19) to achieve a formulation of the error after the weight update:

$$\mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k) = \mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k-1) - e(k)\mathbf{P}(k)\mathbf{\Phi}(k) \tag{4.2}$$

$$\mathbf{\Phi}(k)^T\mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k) = e(k) + \mathbf{o}_{\text{target}} - e(k)\mathbf{\Phi}(k)^T\mathbf{P}(k)\mathbf{\Phi}(k) \tag{4.3}$$

$$\mathbf{\Phi}(k)^T\mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k) - \mathbf{o}_{\text{target}} = e(k) - e(k)\mathbf{\Phi}(k)^T\mathbf{P}(k)\mathbf{\Phi}(k) \tag{4.4}$$

$$e_+(k) = e(k)(1 - \mathbf{\Phi}(k)^T\mathbf{P}(k)\mathbf{\Phi}(k)), \tag{4.5}$$

where $e_+(k)$ represents the output error after the weight update. As mentioned before in Section 2.1.5.4, $\mathbf{P}$ can be written as:

$$\mathbf{P} = \left(\sum_k \mathbf{\Phi}(k)\mathbf{\Phi}^T(k) + \alpha\mathbf{I}\right)^{-1}. \tag{4.6}$$

Due to the used tanh-nonlinearity in Equation (2.3) and the initialization of $\mathbf{\Phi}(k)$ we know that $|\mathbf{\Phi}(k)| \leq \mathbf{1}$. As a result, $\mathbf{\Phi}^T(k)\mathbf{P}(k)\mathbf{\Phi}(k)$ in Equation (4.5) will change from a value close to 1 to a value that asymptotically converges to 0. Consequently, $e_+(k)$ will become small and will eventually converge to $e(k)$. At this point $\mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k) - \mathbf{W}_{\mathbf{r}}^{\mathbf{o}}(k-1)$ becomes 0.

An RNN is infamously difficult to analyze because of its complex dynamics. Despite the efforts made in Tyukin et al. (2003) and Barabanov and Prokhorov (2003), no further progress has been achieved in the quest for rigorous performance and stability guarantees. In this work, however, the following observations concerning stability can be made:

4.2.1 BIBO-stability

Bounded-input-bounded-output stability is guaranteed. The non-linearity in Equation (2.3) (e.g. $\tanh(\cdot)$) and the introduced limiter depicted in Figure 4.3 ensures that the network output is bounded for all inputs.

4.2.2 Local stability

Under certain conditions, local stability at the origin can be guaranteed. The NLq-framework presented in Suykens et al. (1997) is used to derive conditions for local stability of the control system. Before we can apply this framework, we assume that $\gamma = 1$ in Equation (2.3). It is also assumed that learning has converged because a constant change in output weights would make it hard to analyze stability. Under these assumptions we only need to take Network B into account. Furthermore, one needs to ensure that the applied non-linearity $y = f(x)$ fulfills the condition that for each x there exists an $h \in [0,1]$ such that $f(x) = hx$. The applied $\tanh(\cdot)$ satisfies

this condition.

In this chapter I use the NLq framework with $q = 2$ layers where the plant is represented by a neural network interacting in a closed loop with the controller (also a neural network). By preserving the notation used in Suykens et al. (1997) the plant and control network are defined as $\mathcal{M}_1$ and $\mathcal{C}_2$, respectively. In order to simplify the derivation, the use of output bias is omitted so that $\mathbf{a}(k) = \mathbf{\Phi}(k)$ in Equation (2.8). This simplification, however, does not affect the resulting conclusions. According to a discrete version of the notation used in Figure 4.1(b) and Figure 2.1 both networks, one for the plant and one for the controller, can be described by their neural state space models:

$$\mathcal{M}_1 : \begin{cases} \mathbf{b}(k+1) = \tanh(\hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{r}}\mathbf{b}(k) + \hat{\mathbf{W}}_{\mathbf{i}}^{\mathbf{r}}x(k) + \hat{\mathbf{W}}_{\mathbf{b}}^{\mathbf{r}}) \\ y(k) \quad\quad = \hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{o}}\mathbf{b}(k) + \hat{\mathbf{W}}_{\mathbf{b}}^{\mathbf{o}} \end{cases}$$

$$\mathcal{C}_2 : \begin{cases} \mathbf{a}(k+1) = \tanh(\mathbf{W}_{\mathbf{r}}^{\mathbf{r}}\mathbf{a}(k) + \mathbf{W}_{\mathbf{i}}^{\mathbf{r}}y(k) + \mathbf{W}_{\mathbf{i2}}^{\mathbf{r}}y_d(k) + \mathbf{W}_{\mathbf{b}}^{\mathbf{r}}) \\ x[k] \quad\quad = \tanh(\mathbf{W}_{\mathbf{r}}^{\mathbf{o}}\mathbf{a}(k) + \mathbf{W}_{\mathbf{b}}^{\mathbf{o}}) \end{cases}$$

where the network weights and states of $\mathcal{M}_1$ are represented by $\hat{\mathbf{W}}_*^{\Delta}$ and $\mathbf{b}(k)$, respectively. The output weights $\hat{\mathbf{W}}_r^o$ are trained with RLS, the other weights are randomly initialized. $y_d(k)$ denotes the desired plant output. Notice that for $x(k)$ in $\mathcal{C}_2$ the limiter in Figure 4.3 is represented by the $\tanh(\cdot)$-function. After augmenting the states with $\xi(k+1) = x(k+1)$ and substituting the output/input of $\mathcal{M}_1$ with the input/output of $\mathcal{C}_2$, one can write the state space model of the entire control loop as:

$$p_{k+1} = \Gamma_1(V_1\Gamma_2(V_2p_k + B_2w_k) + B_1w_k), \tag{4.7}$$

with $p_{k+1} = [\mathbf{b}(k+1), \mathbf{a}(k+1), \xi(k+1)]^T$, $w_k = [y_d(k), 0, 1]^T$ and both Γ_1 and Γ_2 are matrix representations of the $\tanh(\cdot)$-functions. Here I applied the same notations as in Suykens et al. (1997). The autonomous case is investigated where no external input ($w_k = 0$) to the control loop is considered. Due to the used state space model representation, local stability at the origin is guaranteed if:

$$\rho(V_1V_2) < 1, \tag{4.8}$$

with $\rho(\cdot)$ the spectral radius and V_1V_2 given by:

$$V_1V_2 = \begin{pmatrix} \hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{r}} & 0 & \hat{\mathbf{W}}_{\mathbf{i}}^{\mathbf{r}} \\ \mathbf{W}_{\mathbf{i}}^{\mathbf{r}}\hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{o}} & \mathbf{W}_{\mathbf{r}}^{\mathbf{r}} & 0 \\ \mathbf{W}_{\mathbf{r}}^{\mathbf{o}}\mathbf{W}_{\mathbf{i}}^{\mathbf{r}}\hat{\mathbf{W}}_{\mathbf{r}}^{\mathbf{o}} & \mathbf{W}_{\mathbf{r}}^{\mathbf{o}}\mathbf{W}_{\mathbf{r}}^{\mathbf{r}} & 0 \end{pmatrix}. \tag{4.9}$$

For instance, numerical evaluation of Equation (4.8) on the pitch control task in Section 4.3 gives $\rho(V_1V_2) = 0.8925$, which implies that local stability at the origin of Equation (4.7) is guaranteed in this case. The size of the basin of attraction in

which the controller is locally stable can be large. However as described in Suykens et al. (2000), the basin size in which local stability is proven, is equal or smaller. This basin size can be calculated by maximizing the volume of ellipsoids defined by a quadratic Lyapunov function with respect to p_k. The corresponding matrix inequalities, which constrains this sequential quadratic programming problem, can be found in Suykens et al. (2000). Due to the plant dependence, the attraction basin in which local asymptotic stability is proven, needs to be calculated for each control task at hand, which is beyond the scope of this chapter. I refer to Suykens et al. (2000), for a more extensive description on how this can be calculated.

4.3 Computer simulations and experimental results

To validate the designed controller a number of tasks will be addressed, each with different properties. Applying a model based controller to all these tasks is therefore impossible. In the following experiments it is demonstrated that our IMA control strategy can control different plants with different dynamics even without a predefined internal model. In all tasks where a simulation model is used to evaluate the IMA controller, the control signal is kept constant during the integration time Δt of the simulation model. However, when controlling a real-life physical plant, real-time constraints need to be met. In that case, the IMA controller's calculation time, needed to produce a control command, becomes important.

4.3.1 Inverse kinematics of iCub arm

I mentioned before that, when a robotic arm needs to manipulate objects in its task space, the task performance heavily depends on the used kinematic or dynamic model of the manipulator. Therefore, as first control task, the learning of an inverse kinematic model of the iCub robot arm, shown in Figure 4.5, is considered. The iCub is a humanoid robot developed at the Italian Institute of Technology as part of the EU project RobotCup. It has in total 53 motors for the whole body actuation. In this task, however, the number of degrees of freedom is limited to 7 or one entire arm, without the actuation in the hand. Encoders in each joint measure the angular positions. When a joint position is given to the robot, an internal PID controller will generate the torque necessary to move the joint to the desired position. Therefore, due to the dynamics of the robot, a delay between the commanded and recorded position is observed.

Table 4.1: Joint limits of the iCub robot arm

	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
Min (degree)	-95	0	-37	5.5	-90	-90	-20
Max (degree)	10	160	80	106	90	0	40

4.3.1.1 Model

A Webots simulation model of the iCub robot is used to do the experiments. Webots uses the Open Dynamics Engine for simulating rigid body dynamics. Instead of using a known forward kinematic model of the manipulator to determine the current end-effector position, an external supervisor is used that can accurately measure the absolute coordinates of the end-effector according to the robot's frame of reference. The actuation of each joint is limited to a specific range, shown in Table 4.1. Within this range it is, however, still possible to have collisions between the robot arm and other parts of the robot. The default Webots integration time step $\Delta t = 32$ ms is used for the task.

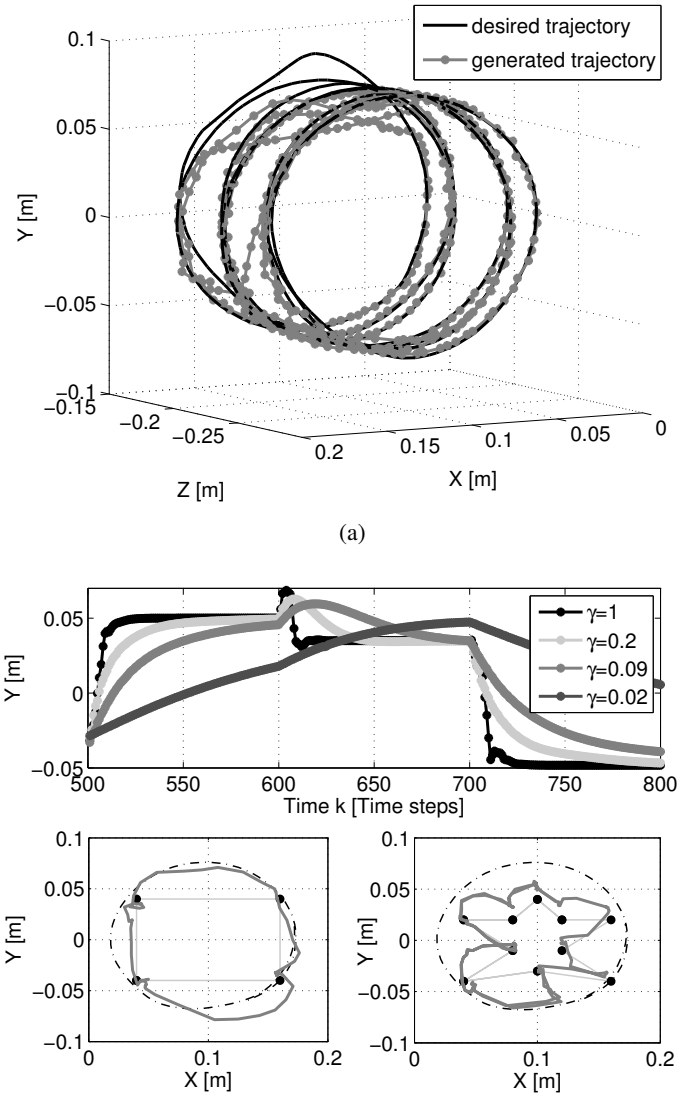
4.3.1.2 Controller

To learn the inverse kinematics of the iCub's 7 DOF robot arm a Reservoir Computing network is used as basic module in the IMA control framework. The introduced limiter bounds the generated joint angles $\mathbf{x}(t) = [\theta_1(t), \theta_2(t), ..., \theta_7(t)]^T$ to the joint limits introduced in the previous section. These bounded values are used to train the output of network A which is driven by the corresponding end-effector position measured by the supervisor ($\mathbf{y}(t) = [X(t), Y(t), Z(t)]^T$). All these values are first normalized in such a way that the standard deviation of both the network inputs and outputs are equal to 1, as discussed in Chapter 2. The network parameters, shown in Table 4.2 are sensible defaults but not optimized. It is however possible to find a better set of parameters by performing for instance a grid search on a validation set. Parameters which are not mentioned such as f_o^r are set to 0.

The used RLS-parameter α, defined in Section 2.1.5.4, is set to 1. The initial output weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}(k)$ are normalized random values ($\mathcal{N}(0,1)$).

4.3.1.3 Results

The desired spiral trajectory in task space that the controller should learn to follow is similar to the one described in Reinhart and Steil (2009). By connecting the IMA controller to the iCub simulation model, the controller will initially drive the robot arm randomly. Thanks to the RLS learning rule, fast adaptation of the output



(a)

(b)

Figure 4.4: (a) This plot illustrates a generated trajectory after training (gray) together with the desired trajectory (black). The dots on the generated trajectory are the sample points. (b, top) Demonstrates the velocity modulation on target point reaching (100 time steps per target point) . (b, bottom) Generalization (dark gray) to different target points (black dots). The circular dashed line represents the projection of the spiral on the plain of the target points.

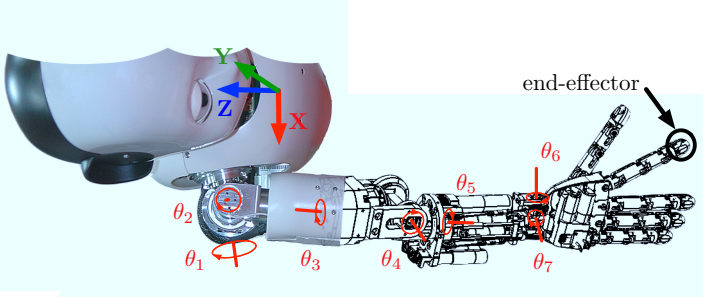


Figure 4.5: The control task: Which angles θ_i are necessary to achieve a desired pitch end-effector position $[X,Y,Z]^T$?

Table 4.2: Network parameters of the inverse kinematics learning task

Parameter	Value	Parameter	Value
N	400 neurons	f_i^r	1.5
ρ	1	f_b^r	0.5
δ	1	γ	1

weights is achieved. After only 1000 time steps (time step = 270 ms) the robot starts following the spiral trajectory. Most feedback controllers use an error defined in the task space to achieve the desired behavior. Although the IMA controller is not designed to directly minimize this error in task space, it converges to the desired trajectory because the trained model corresponds to that of the iCub's arm. When such a model is achieved one could choose to continue the training online, learning the inverse kinematics in newly visited task space regions. However, to evaluate the trained model at a certain point in time, the training is stopped by setting $\Delta \mathbf{W}_{\mathrm{r}}^{\mathrm{o}} = \mathbf{0}$ in (2.19) and evaluate how well it can follow a desired trajectory without learning. In Figure 4.4(a), such generated trajectory of the iCub's arm (gray) is shown, which needs to follow a novel spiral trajectory. As demonstrated, the learned inverse kinematic model corresponds well with the iCub's arm for the desired end effector positions. However, because of the physical limitations of the robot, some desired trajectory points, especially the ones closer to the robot ($Z < -0.2$), are unreachable by the robot.

Next, the transient and generalization behavior of the learned kinematic model is investigated. Instead of following a spiral trajectory, a number of target corner points that form a specific shape (e.g.: a square or a star) are defined. These points are all located on a plane perpendicular to the axial direction of the spiral. Each target point excites the network for 100 time steps. Afterwards, the next target point is used as desired position. In other words, it is not necessary to follow a square or star trajectory, but the task is to reach the target corner points of each shape. The target points forming a square are located on the projection of the learned spiral data (dashed lines). As shown in Figure 4.4(b) (bottom, left) the desired target points are reached. However, the generated movement between two different target points is demonstrating transient behavior, that is, it does not follow a straight line (shortest path between target points) but rather according to an arc (dark gray). The acquired kinematic model was learned by following sampled points of a spiral trajectory, never reaching other regions of the task space, which explains the transient arc behavior of the generated motion. The target points forming a star shape are, except for two, not located on the projection of the spiral trajectory. Although the learned kinematic model is based on the data seen while following the spiral trajectory, the model is generalizing quite well to the other target points. Figure 4.4(b) (bottom, right) shows small deviations in the reached target points (black dots).

The velocity at which each trajectory point is reached can be modulated after or during training. This is achieved by changing the leak rate γ of the reservoir states (2.3). As described before, changing $\gamma \in [0,1]$ effectively changes the time scale of the system. Figure 4.4(b) (top) demonstrates the effect of such modulation for multiple γ's after learning ($\Delta \mathbf{W}_{\mathrm{r}}^{\mathrm{o}}(k) = \mathbf{0}$) and for different target points (e.g.: the target points forming the star shape). By decreasing γ, the distance between each sample point will decrease as well. As shown in the top plot of Figure 4.4(b),

the robot is unable to reach the target positions within 100 time steps when using $\gamma = 0.02$.

4.3.2 Heating tank temperature control

The second control task is a process with a variable dead-time and slow nonlinear dynamics. These control problems appear in industrial processes where measurement sensors that are used for feedback, are not integrated in the process itself (e.g. solar collector field (Torrico et al., 2010)). In this experiment, as shown in Figure 4.6, the system consists of a filled and constantly heated water tank with attached pipe of length L. If the output temperature of the pipe is controlled by the throughput of water that feeds the tank, this temperature depends not only on the pipe length (L) but also on the throughput itself, which is a control parameter. The fact that this parameter constantly changes (leading to a variable dead time) has a significant impact on the performance of the control loop. Controlling such a process is a challenging task, especially without an instantaneous measurement of the process variables or with control by a delayed pump (controlling the throughput) (Richard, 2003), where the response on the feedback is delayed.

4.3.2.1 Model

The dynamics of the plant model illustrated in Figure 4.6 are described by the following nonlinear differential equation:

$$\rho c_p V_{tank} \frac{dT_{tank}(t)}{dt} = Q + \rho c_p q(t)(T_{in} - T_{tank}(t)), \tag{4.10}$$

where c_p denotes the specific heating capacity of water, V_{tank} the volume of the tank, $T_{tank}(t)$ the water temperature in the tank, T_{in} the temperature of the added water, Q the added heat, ρ the density of water and $q(t)$ the throughput of the added water. The dynamics of the outlet pipe with length L and area S are modeled by the following low-pass filter:

$$\frac{T_{tube}(s)}{T_{tank}(s)} = \frac{K_{tube}}{T_{tube}s + 1}, \tag{4.11}$$

where K_{tube} is the fraction of temperature change from tank to tube and T_{tube} an unmeasurable temperature with temperature T_{out} that follows the equation:

$$T_{out}(t) = T_{tube}(t - d(t)). \tag{4.12}$$

In the previous equation $d(t)$ describes the variable dead-time which equals:

$$d(t) = T_s N_d = \frac{LS}{q(t)}, \tag{4.13}$$

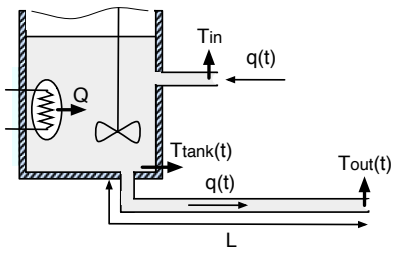


Figure 4.6: Illustration of the heating tank process. A tank filled with water is constantly heated with heat flow Q. Water with temperature T_{in} is pumped in and exits the tank with a throughput $q(t)$. The water that is leaking from the tank has a certain amount of time to cool down in a pipe with length L before measurement of temperature T_{out}. To insure homogeneous heating, the water in the tank is stirred. The control task: How to change the throughput $q(t)$ to get a desired temperature T_{out}?

where T_s represents the sampling period. N_d describes the unknown dead-time. It is clear that by knowing L, S and $q(t)$ the variable dead-time can be calculated. During simulation the parameters given in Table 4.3 were used. For simulation, the Dormand-Prince method (Dormand and Prince, 1980), also known as Runga-Kutta (4,5), with an integration time step Δt of 4 s was employed.

4.3.2.2 Controller

To control this plant I use a ESN in the IMA control framework described in Section 4.1.1. The limiter bounds the throughput $\mathbf{x}(t) = [q(t)]$ to the allowed values for $q(t)$ shown in Table 4.3. Next, the output connections of network A are trained

Table 4.3: Simulation parameters for the heating tank model

Parameter	Value	Parameter	Value
c_p	4186 J/kgK	K_{tube}	0.99
$V_{tank} = LS$	1.13 l	T_s	4 s
Q	1100 J	$q(0)$	0.0167 l/s
ρ	1 kg/l	$q(t)$	$\in [0.005, 0.03]$ l/s
T_{in}	15 °C		

Table 4.4: Network parameters of heating tank task

Parameter	Value	Parameter	Value
N	500 neurons	f_i^r	0.1
ρ	1	f_b^r	0.5
δ	30	γ	0.5

with the values $\tilde{\mathbf{x}}(t-\delta\Delta t)$. The feedback values $\mathbf{y}(t) = [T_{out}(t)]$ from the plant are given to the networks in a normalized form (subtracted with the mean an divided by its standard deviation). The parameters of both networks, shown in Table 4.4, were optimized by performing grid search on a validation set (target temperatures forming a staircase signal).

The introduced RLS-parameters defined in Section 2.1.5.4 are set to $\lambda = 1 - 10^{-6}$ and $\alpha = 10$. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

4.3.2.3 Results

The controller is applied to the described simulation model for 12000 time steps or 13.33 hours real time. The desired response of the plant consists of different phases where, in the first phase, the framework tries to control the plant to have a $y(t)$ that changes relatively quickly. In this phase, red noise is used by feeding white noise through a low-pass filter. Afterwards, this noise is scaled to represent realistic temperature values. The second phase consists of a staircase signal. Both phases are randomly generated for each experiment. The first 6000 time steps of the experiment are shown in Figure 4.7. One can see, by looking at the average quadratic change in output weights, that the IMA controller is learning to control the plant within the first 2000 time steps. In Figure 4.8 the transition to a staircase signal is shown. Here the controller is able to adapt by changing its output weights according to the desired plant output. As shown at the bottom of both Figure 4.7 and 4.8, the generated throughput during this staircase signal, is close to an optimal control signal. Indeed, a temperature decrease is generated by setting the throughput very high in the beginning and lowering it afterwards.

I compared the IMA controller with a model based controller, called the Nonlinear Predictive Control Strategy (NEPSAC), that out-preforms more classical approaches (such as PID) on this task (Gálvez-Carrillo et al., 2009; De Keyser, 2003). For comparison, a staircase signal is used as desired plant output which, after 2000 time steps of initialization, is shown in Figure 4.11.

We notice that both implementations of the NEPSAC and the IMA controller have some trouble in the beginning due to the transition between the faster variation

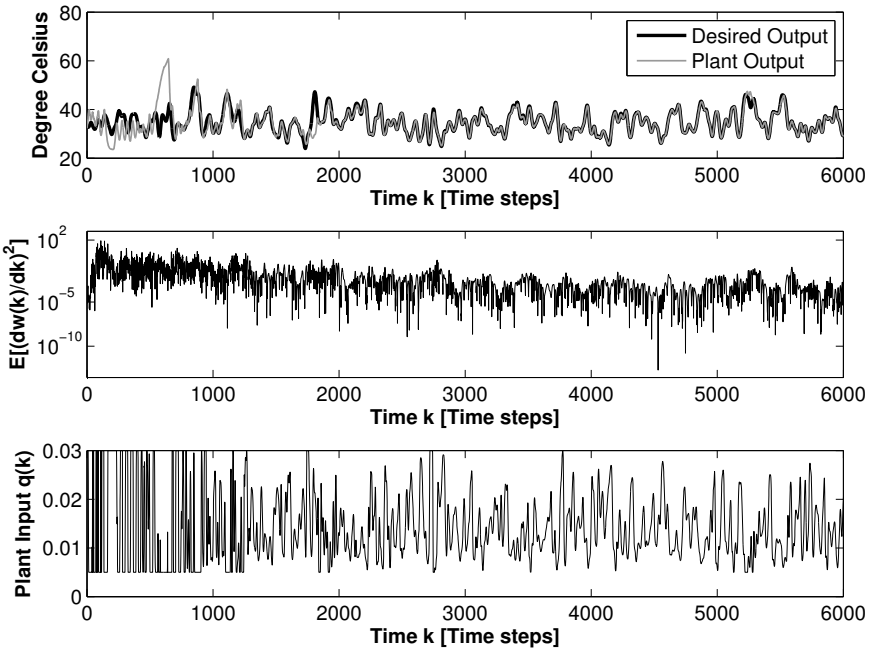


Figure 4.7: Overview of the first 6000 time steps of the simulation. Here, the desired output is an always changing temperature. Above, the actual plant-output (shifted over δ time steps) together with the desired one are shown. In the middle, the average quadratic weight adaptation is illustrated. At the bottom, the actual plant-input, which is generated by the controller, is shown.

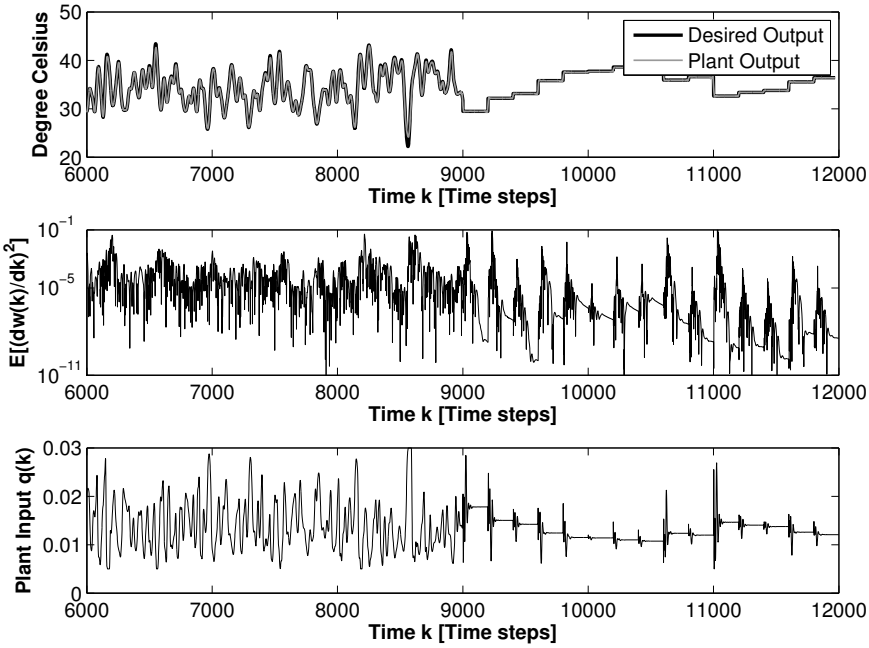


Figure 4.8: Overview of the last 6000 simulation time steps. Here, the desired output is a constantly variating temperature, that eventually shifts to a desired temperature profile according to a staircase. The middle plot illustrates the average quadratic weight adaptation of the output weights. The bottom plot shows the generated control signal.

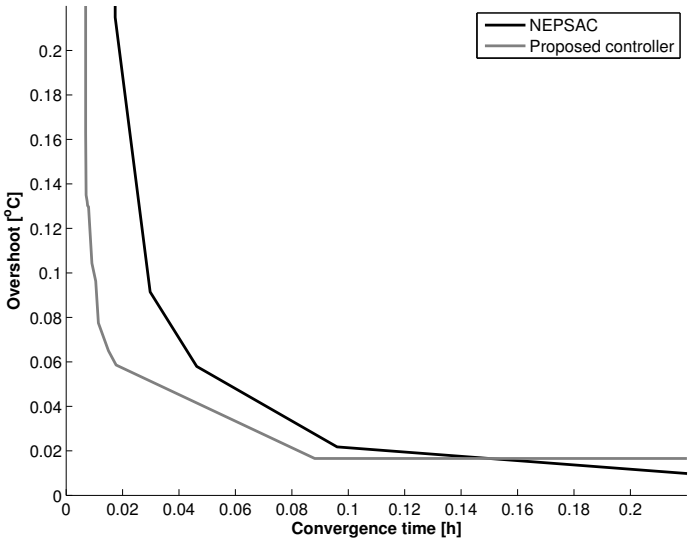


Figure 4.9: This plot shows the Pareto front for both the IMA and the NEPSAC controller. When the overshoot is allowed to be equal or larger than $0.016°C$, the proposed IMA controller is preferred. For a smaller overshoot but a larger convergence time NEPSAC is better in the comparison.

in output temperature and the staircase signal. Afterwards, both are able to follow the desired temperature. Taking a closer look at the staircase in Figure 4.12 between time steps 5000 and 5800 reveals that, after a temperature change, the IMA controller is able to reach the desired temperature faster than NEPSAC. The time to reach the desired output temperature is called converging time.

The convergence time is evaluated as the time needed to approach the set point after which it stays within a predefined margin around this set point. In the following experiments for this control task I have set this margin to $0.01°$C. The overshoot is measured as being the largest difference between the desired set point and the produced plant output after the set point has changed.

For NEPSAC the balance between overshoot and convergence time is regulated by its prediction horizon $\in [5,\ldots,50]$ which is illustrated by its Pareto front (Horn et al., 1994) in Figure 4.9. The larger its prediction horizon the smaller the overshoot but with the disadvantage that the convergence time increases.

As stated in Section 4.1.1, δ defines a time window with which the dynamics of the plant are observed. If δ is small, the learned model is more sensitive to fast dynamical changes, and vice versa. The used leak rate on the other hand, basically implements a low-pass filter on the state changes. As a results, the

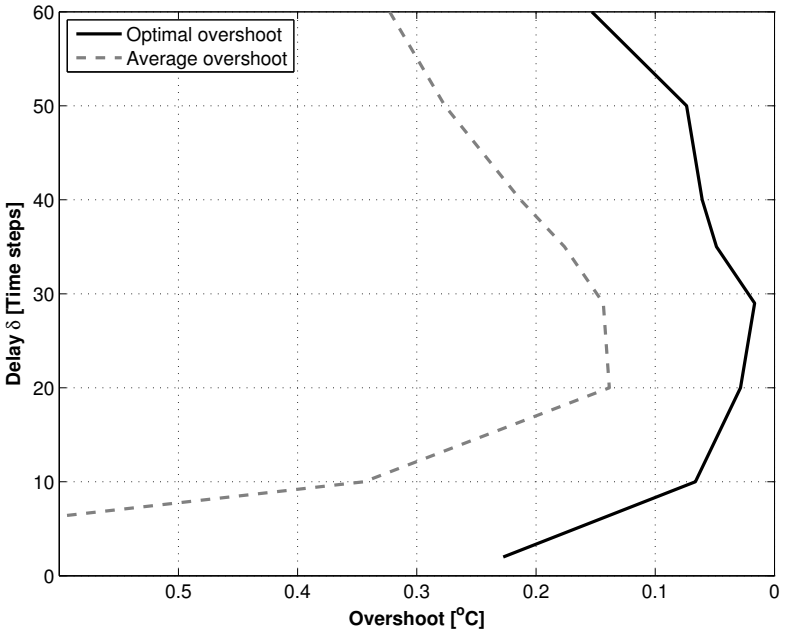


Figure 4.10: Illustration of the effect of the delay $\delta\Delta t$ on the optimal and average overshoot of the IMA controller. The average overshoot is calculated over all experiments with the same delay $\delta\Delta t$ but with different leak rates and input scaling. Increasing δ will improve the convergence time until the delay becomes larger than the memory capacity of the ESN. The memory capacity of an ESN with 500 neurons starts to decrease around 30 time steps after which it will decrease dramatically.

overshoot/convergence-time balance for the IMA controller is depending on both the value of the delay δ and the leak rate γ. In Figure 4.9 the Pareto front for the optimal $\delta = 29$ and averaged over 5 reservoirs is given. This means that the defined balance is controlled by its leak rate and input scaling. For a defined input scaling, increasing the leak rate will lower the overshoot and increases the convergence time. However, experiments show that the choice of the delay $\delta\Delta t$ influences the resulting Pareto front as well. Not only is δ depending on the rate at which relevant samples are presented to the network but also on the memory capacity of the network itself. This is shown in Figure 4.10. For different delays the optimal/lowest and average[1] overshoot is shown. One can notice the improvement in overshoot by increasing the delay δ until the delay becomes larger than the memory capacity[2] of the network (around 30 time steps for a reservoir with 500 neurons). Increasing the delay δ further will lead to a larger overshoot.

Now, if we compare both Pareto fronts one can conclude that the IMA controller is more suitable for tasks where fast convergence is needed and the overshoot is allowed to be larger than $0.016°$C. For slower control where the desired overshoot is lower and a convergence time of 0.15 hours or larger is allowed, NEP-SAC is the optimal choice.

4.3.3 Aircraft pitch control

The third task that is considered is taken from a set of control examples (Messner and Tilbury, 1996). The purpose of this task is to control the pitch of a simplified aircraft model by changing the elevator deflection angle. However, changing this deflection angle causes the pitch angle to move slowly. The time needed to reach the desired angle depends on the distance between the previous and current pitch angle which makes this task nontrivial.

4.3.3.1 Model

The pitch control problem is simplified by assuming a steady cruise of the aircraft at constant velocity V and altitude. Under these conditions the control problem can be formulated as:

$$\frac{d\alpha}{dt} = -0.313\alpha + 56.7q + 0.232\eta$$

$$\frac{dq}{dt} = -0.0139\alpha - 0.426q + 0.0203\eta$$

[1] Average over all experiments with the same δ but with different leak rates and input scaling.

[2] The number of time steps, traces of past inputs are reflected in the current states (length of short-term memory).

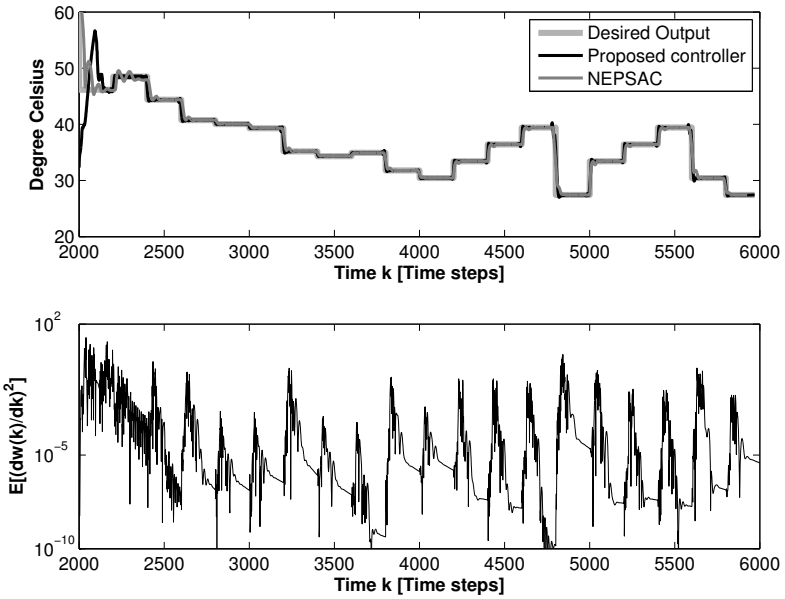


Figure 4.11: The actual plant output is shown for the IMA controller (shifted over $\delta\Delta t$) and for the NEPSAC-controller. This gives, together with the desired plant output, a good representation of the control performance.

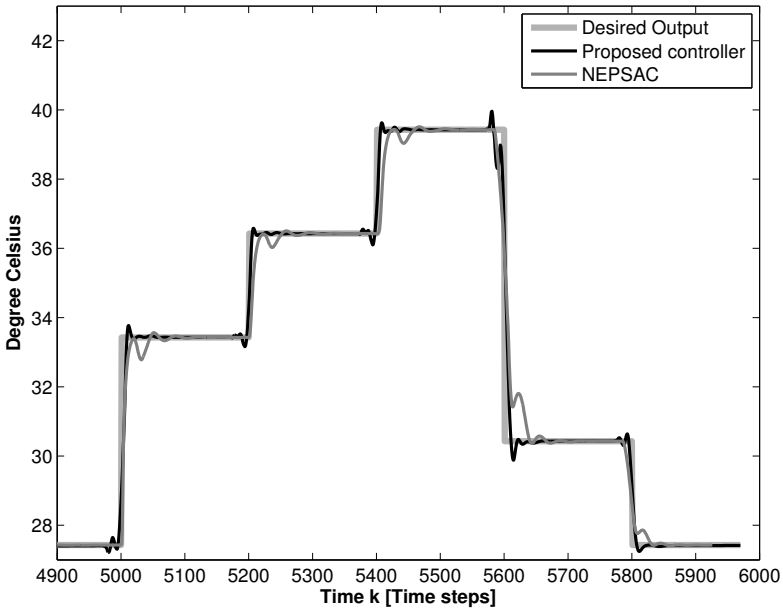


Figure 4.12: This plots shows a more detailed view of the overshoot and convergence time of both controllers.

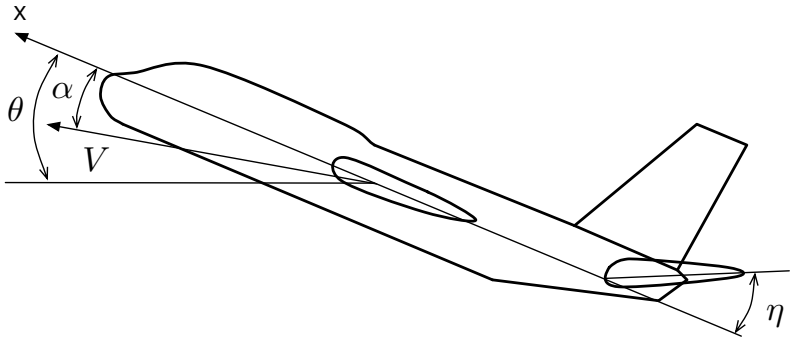


Figure 4.13: The representation of the aircraft which is used in simulation. V represents the velocity vector and X denotes the center axis of the plain. Θ, α and η on the other hand represent the pitch, attack and elevator deflection angle, respectively. The control task: How to control η to achieve a desired pitch angle Θ?

Table 4.5: Network parameters of pitch control task

Parameter	Value	Parameter	Value
N	500 neurons	f_i^r	0.2
ρ	0.7	f_b^r	1
δ	10	γ	1

$$\frac{d\Theta}{dt} = 56.7q.$$

As shown in Figure 4.13, α is describing the angle of attack, q the pitch rate, η the elevator deflection angle and Θ the pitch angle. For simulation I again use the Runga-Kutta (4,5) method with an integration time step of 50 ms.

4.3.3.2 Controller

The dynamics of the simplified control task are linear. However, they present some interesting challenges for the proposed learning algorithm. Changing the elevator angle η causes the angle of attack α to change slowly until it settles. When η is changed again before α has settled, the current angle of attack is partially depending on its previous angle. The pitch of the aircraft is controlled by changing the elevator deflection angle. Therefore, the IMA controller, shown in Figure 4.3, has an input and output which is defined as $y = \Theta$ and $x = \eta$, respectively. In Table 4.5 the network parameters for both network A and B are shown. All these parameters were determined by performing a grid search on a validation set.

4.3.3.3 Results

For the evaluation of the controller some experiments are conducted where a desired pitch angle is set. After keeping the target pitch angle constant for 300 time steps, the target pitch angle is changed to another value. These values are randomly chosen according to a standard normal distribution: $y_d(t) \in \mathcal{N}(0, 0.35)$ rad. Each experiment takes 10000 time steps. To evaluate the control performance it is compared with a method called Linear Quadratic Regulator (LQR) (Kwakernaak and Sivan, 1972; Sontag, 1998) which tries to minimize the following cost function:

$$J = \sum_{k=0}^{K} \left(\chi(k)^T \mathbf{Q} \chi(k) + \eta(k)^T \mathbf{R} \eta(k) \right), \tag{4.14}$$

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & p \end{bmatrix} \tag{4.15}$$

$$\mathbf{R} = 1 \tag{4.16}$$

This method allows us to find (tuning a weighting factor p) an optimal control matrix $\mathbf{K}$ that results in an appropriate linear state-feedback controller $\eta = -\mathbf{K}\chi$ (χ represents the controller state $[\alpha, q, \Theta]^T$). In Messner and Tilbury (1996) and Hafner and Riedmiller (2011) an LQR controller design is presented which is used here as well and which results in a gain vector $\mathbf{K} = [-0.6435, 169.6950, 7.0711]$ with a weighting factor $p = 50$ that weights the importance of the error compared to the importance of the control effort. The calculation of $\mathbf{K}$ uses a Ricatti equation, similar to RLS and the Kalman filter. In fact there exists a duality between the LQR (controllability) and the Kalman filter (observability). The $\mathbf{Q}$ matrix and R used correspond to respectively the variance of the process noise and the variance of the measurement noise. In optimal control the Linear Quadratic Gaussian (LQG) controller is one of the most fundamental controllers. It combines the strength of both the Kalman filter for observability and the LQR for controllability.

As shown in Figure 4.14, the IMA controller's performance improves as the experiment progresses. The learned controller is changing the deflector angle $x = \eta$ fast after a set point adjustment. Afterwards, as the pitch angle $y = \Theta$ converges, the generated output converges to 0 rad. In Figure 4.15 a more detailed section of such an experiment is shown ($\delta = 3$ and $\gamma = 0.95$). Here, the difference between both controllers and the desired plant output is clearly visible. The learned controller causes the pitch angle to change rather fast before approaching the desired set point. After producing a very small overshoot, the resulting pitch angle converges. This small overshoot is clearly less than the overshoot of the LQR approach. However, as in most control tasks, a trade-off between overshoot and convergence time has to be made. To ensure a good comparison of both metrics the design requirements used to design the LQR-controller should be the same as the ones used for the IMA controller. However, by creating a Pareto front of both approaches one can evaluate and compare the control performance more thoroughly.

As in the previous task the overshoot is calculated but use a margin of 0.0005 rad to determine the convergence time.

The overshoot and convergence time of the IMA controller was calculated for different parameters values of $\gamma \in [0.6, \ldots, 1]$ and $\delta \in [2, \ldots, 12]$. A large γ results in a smaller overshoot than a smaller γ. Consequently, the convergence time will be larger with a large γ than with a small γ. Similarly as for the IMA controller, the overshoot and convergence time of the LQR approach were calculated for multiple weighting factors $p \in [3, \ldots, 150]$. These experiments result in the Pareto front shown in Figure 4.16.

The Pareto front illustrates that the IMA controller is performing worse than LQR when a small convergence time is needed (< 6.5 s). However, when a larger convergence time is allowed the resulting overshoot of our controller is much smaller (Figure 4.15). Therefore, under these conditions, the presented controller is more appropriate than the LQR approach.

The Pareto front of the LQR approach is determined by performing a parameter sweep of the weighting factor p. As the LQR approach is fully deterministic, the results are all located on the Pareto front.

With LQR a small overshoot and a lower convergence time is possible for smaller weighting factors ($p < 100$). The Pareto front of the proposed IMA controller is calculated by averaging results over 6 RC-networks.

4.3.4 Balancing double inverted pendulum

The balancing task of a double inverted pendulum is a well known task in control theory and presents some interesting control challenges. Here, 2 rods connected with a joint need to be balanced in a upright position by only controlling the angle of one of the rods. In a small region around this desired position the dynamics are approximately linear. However, outside this region the dynamics of the pendulum are strongly nonlinear. In this task only the pendulum stabilization is considered and not the swing-up.

4.3.4.1 Model

The double inverted pendulum is modeled as illustrated in Figure 4.17. In this model the weight distribution of the rods is neglected and each end of the rod is modeled as a point mass. The Cartesian coordinates of these point masses are given by (x_1, y_1) for m_1 and (x_2, y_2) for m_2, with:

$$\begin{aligned} x_1 &= l_1 \sin(\theta_1) \\ y_1 &= -l_1 \cos(\theta_1) \\ x_2 &= l_1 \sin(\theta_1) + l_2 \sin(\theta_2) \\ y_2 &= -l_1 \cos(\theta_1) - l_2 \cos(\theta_2). \end{aligned} \tag{4.17}$$

Using these equations the potential energy V and kinetic energy T can be derived:

$$\begin{aligned} V &= m_1 g y_1 + m_2 g y_2 \\ T &= \frac{1}{2} m_1 v_1^2 + \frac{1}{2} m_2 v_2^2, \end{aligned} \tag{4.18}$$

where $v_i = \frac{dx_i}{dt} + \frac{dy_i}{dt}$. To validate the model one can compute the total energy $E = V + T$, which should be a constant over time when the applied torque is zero.

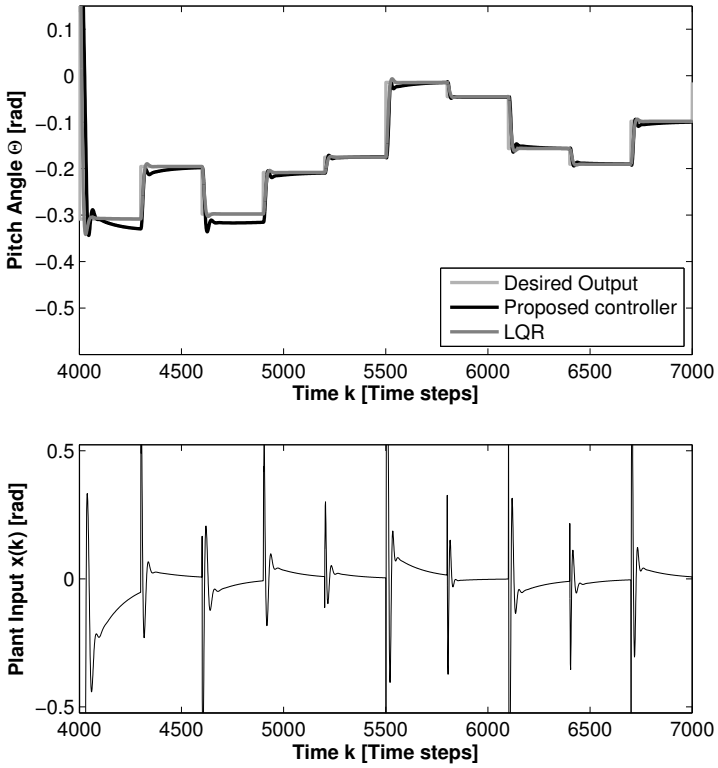


Figure 4.14: The top plot shows a part of a pitch control experiment where the desired pitch angle is compared with the ones acquired by using the proposed IMA controller and the Linear Quadratic Regulator approach. It demonstrates an improvement in the convergence of the IMA controller as the experiments progresses. The bottom plot shows the corresponding plant input (elevator angle η) produced by the IMA controller.

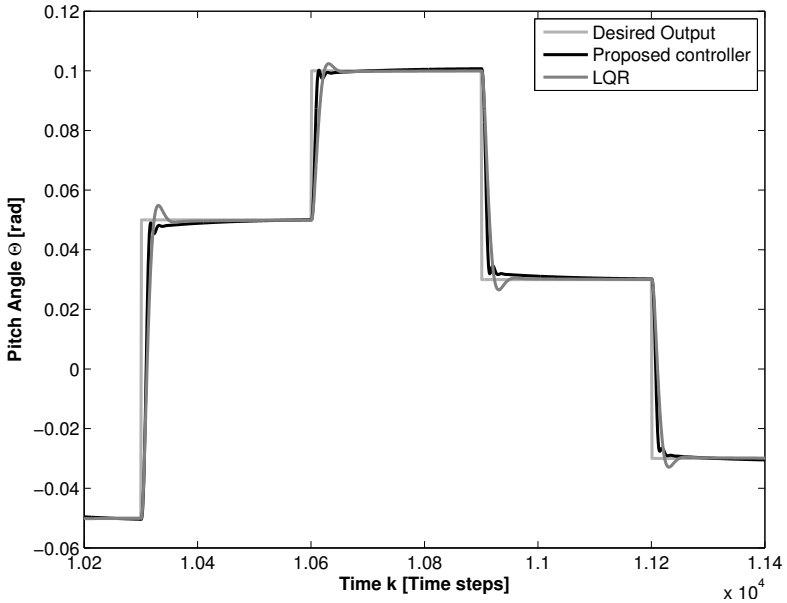


Figure 4.15: This plot gives a detailed view of the proposed IMA controller's performance at the end of an experiment. Furthermore, the plant-output under influence of the LQR approach is shown, which illustrates the difference in convergence time and overshoot between both approaches.

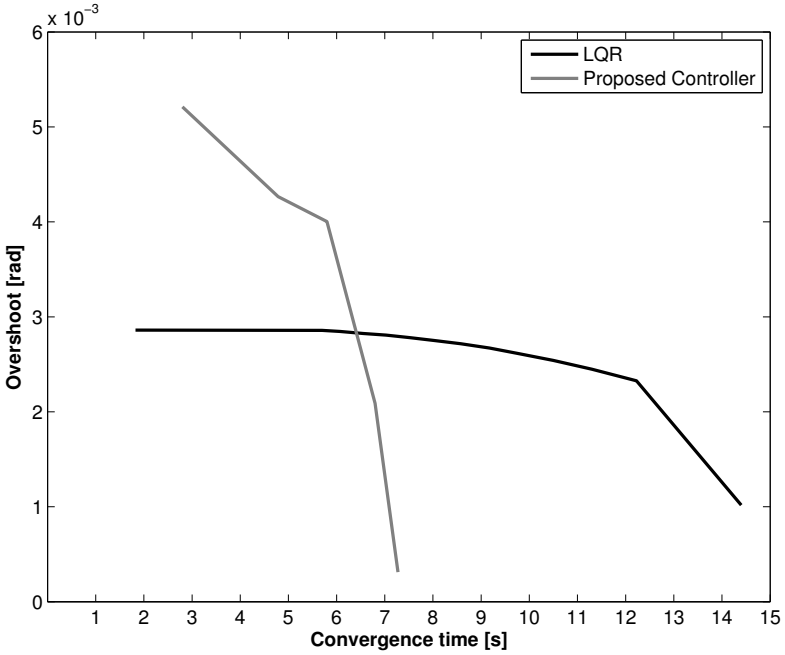


Figure 4.16: This plot shows the Pareto front of both the IMA controller and the LQR-approach, given the predefined margin conditions (error < 0.0005 rad). It illustrates that the LQR-controller performs better as long as a convergence time of less than 6.5 s is required. As soon as a the pitch angle is allowed to converge slower, the proposed IMA controller is recommendable. Furthermore, most of the results of the LQR approach converge fast. Only when the weighting factor p becomes large, the convergence time increases drastically.

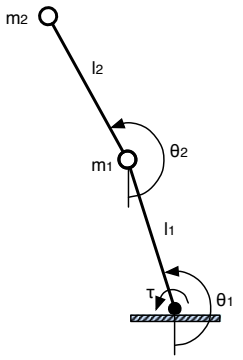


Figure 4.17: This is an illustration of the double pendulum that is used as simulation model. The control task: How to drive the torque of the motor to achieve balancing of the double inverted pendulum?

Next, I use the Lagrangian transformation with $L = T - V$ and define the applied torque τ by using the Euler-Lagrange differential equation:

$$\frac{\partial L}{\partial \theta_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}_i}\right) = \tau_i, \tag{4.19}$$

with $\tau = \tau_1$ and $\tau_2 = 0$ because only torque is applied on the first rod. By writing this equation in function of θ_i and $\frac{d\theta_i}{dt}$ for $i = 1,2$ one can solve this system with respect to $\frac{d\theta_i}{dt}$, which leads to the equations of motion for the double pendulum (Figure 4.17). Similar as in the previous tasks the Runga-Kutta (4,5) method is used with an integration time step Δt op 50 ms.

Table 4.6: Simulation parameters for the double pendulum

Parameter	Value	Parameter	Value
m_1	1 kg	θ_1	$\in [\frac{\pi}{2}, \frac{3\pi}{2}]$
m_2	1 kg	θ_2	$\in [\frac{\pi}{2}, \frac{3\pi}{2}]$
l_1	1 m	$\tau_{\max}$	50 Nm
l_2	1 m	Δt	50 ms

Table 4.7: Network parameters of double inverted pendulum task

Parameter	Value	Parameter	Value
N	300 neurons	f_i^r	1
ρ	1	f_b^r	0.5
δ	1	γ	1

4.3.4.2 Controller

This control task is different from the other tasks in the sense that the IMA controller in this task is limited in the number of examples and the amount of time given to learn. In most cases an initial control effort does not succeed in balancing, which causes the rods to fall down. The amount of information that can be used for learning is therefore limited. As a result, balancing the pendulum means that the range of a possible solution is restricted for θ_1 and θ_2 (shown in Table 4.6). This region consists of both a linear and nonlinear dynamic region of the pendulum. It is assumed that the controller is unable to control the pendulum when the pendulum exceeds the restricted range and, when this happens the simulation is reset. Each balancing trial until a simulation reset, is called an episode. A simulation reset implies randomly initializing both rod positions within $[\pi - 0.15, \pi + 0.15]$, holding this initial position for δ time steps and reinitializing the network states to their original values.

For this task the proposed IMA controller, shown in Figure 4.3, has an input and output, which are defined as $\mathbf{y}(t) = [\theta_1, \theta_2]$ and $\mathbf{x}(t) = [\tau]$, respectively. The angle θ_2 of the second rod is scaled up with an experimentally determined value of 10 by applying grid search. As a results, a small change of θ_2 will have a larger influence on the network than a small change of θ_1. Consequently, the network will first prioritize the stabilization of θ_2 before stabilizing θ_1 to the desired angle. The output of network B is limited by a limiter to $\tilde{\mathbf{x}}(t)$ by insuring $|\mathbf{x}(t)| < \tau_{\max}$. The network parameters used for both networks are shown in Table 4.7. All parameters were optimized by applying a grid search. The introduced RLS-parameters are set to $\lambda = 1 - 10^{-6}$ and $\alpha = 1$. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

4.3.4.3 Results

Each conducted experiment takes 50000 time steps or 41.7 minutes. For each episode, both rods are randomly initialized to a value in $[\pi - 0.15, \pi + 0.15]$.

The top two plots of Figure 4.18, show such an experiment where balancing to

the desired upright position was achieved after 8 episodes. The end of an episode is indicated by a vertical dotted line. We notice an increase in episode duration due to the learning progress of the IMA controller. In the 9th and final episode balancing is achieved. However, this does not imply that the acquired controller will be able to balance the pendulum given any initial position of the rods. In order to evaluate this, the pendulum is again randomly initialized in the range $[\pi - 0.15, \pi + 0.15]$, after successful balancing was achieved. The bottom rows of Figure 4.18, demonstrate the results for two different initializations. For both these experiments, the initial balancing efforts are shown for the first 50 time steps together with an overview of the entire experiment. These experiments clearly demonstrate that the IMA controller is indeed learning a balancing controller that is able to generalize for different initializations.

Although no convergence is achieved is some episodes, they can take a long time because of the good balancing efforts of the controller. As a result, some experiments finish before actual convergence to the upright position emerged. In this evaluation of the controller convergence is assumed when the errors on both angles clearly become smaller in the last episode. The number of episodes needed to achieve balancing averaged over 40 experiments is 13.75 episodes. The minimum

Furthermore, one can notice, due to the chosen input scaling, that the pendulum is indeed balanced by first prioritizing the convergence of θ_2 and afterwards θ_1.

4.3.5 Cart pole balancing

Another well known task that is often used as benchmark is the cart pole balancing problem. Here a single pendulum is mounted on a cart using an unactuated joint and the cart, which glides on a rail, is driven by an external force. This task poses similar control challenges as the previous task. However, an extra difficulty is added by the size of the rail, which limits the range in which the cart can move. As in the previous task only the balancing problem is considered and not the swing-up. The reason for this is that the IMA feedback controller will only try to directly move the pendulum in the desired direction. However, because it is an underactuated system the swing-up can only be achieved by performing actions that cause the pendulum to also move away from the desired direction. Only when enough momentum is achieved by these back and forth movements a swing-up can be accomplished.

4.3.5.1 Model

Instead of using a simulation model as in the previous tasks, the controller is applied on a real life cart pole setup (shown in Figure 4.19(b)) that I have build. The angle of the unactuated joint is measured by a wireless encoder of which the development was initiated during a student project (under the supervision of former lab member Michiel D'Haene and myself) and later extended by me in order to allow multiple

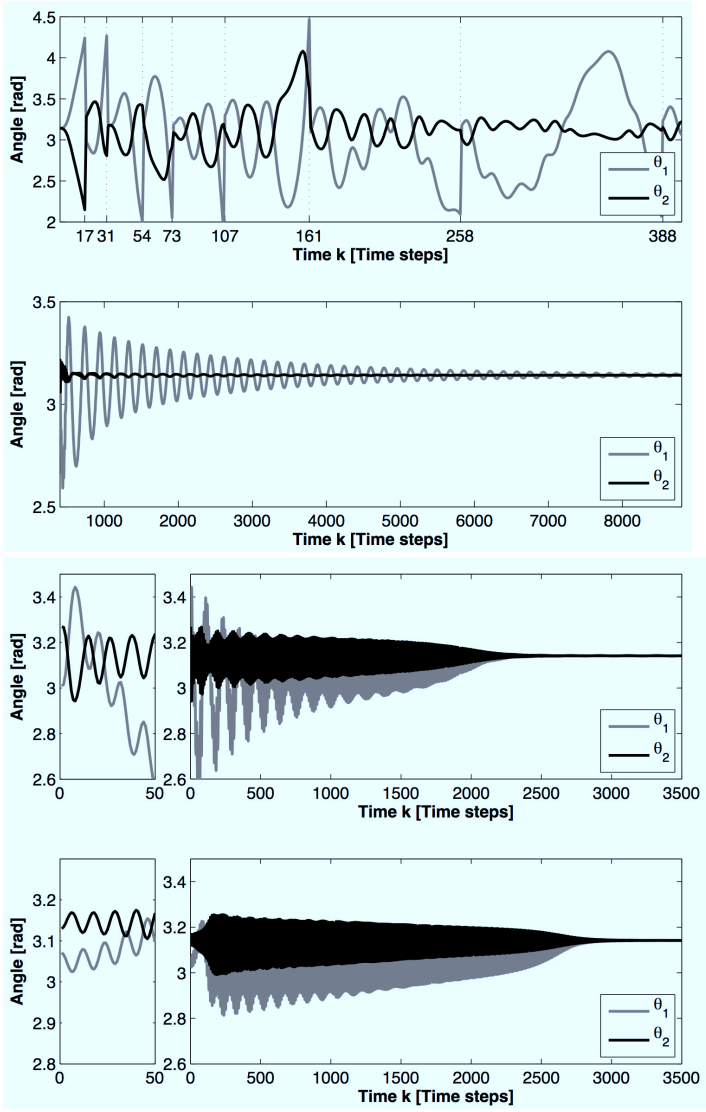
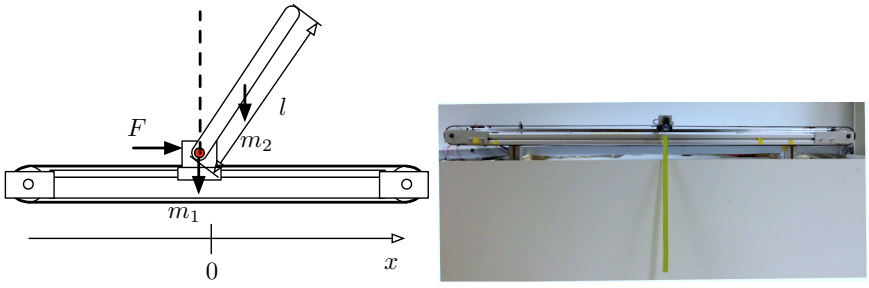


Figure 4.18: Both top plots illustrate the learning progress of the controller in the double inverted pendulum task. The top plot shows the first efforts of trying to balance the pendulum. Each vertical dotted line marks the beginning of a new episode at which the pendulum states are reset. In this experiment 8 episodes are necessary to successfully balance the pendulum in the 9th episode. The continuation of this last episode is shown in the second plot. The two bottom plots each illustrate the effect of initializing the pendulum in a random position after successful balancing the pendulum. For each experiment, both the initial course and an overview of the entire experiment are shown.



(a) (b)

Figure 4.19: (a) An illustration of the real cart pole setup. The red dot indicates the location of the wireless encoder that measures the pendulum's angle. (b) Shows the actual cart pole setup on which the presented experiments were performed. The control task: How to drive the torque of the motor (force on cart) to achieve balancing of the inverted pendulum?

encoders to be used at the same time. The use of this setup implies that no accurate model is available.

4.3.5.2 Controller

Similar to the controller in the previous task, the training needs to happen during consecutive episodes. This means that the data, that is observed by the IMA controller, is limited to the controllers own balancing efforts. Initial control efforts will not succeed in balancing, which causes the pendulum to move downwards easily. Therefore, the experiment is reset each time the cart reaches its limits or when the pendulum angle θ exceeds the predefined range $|\theta| < 60°$. Resetting the experiment on a real setup means that the pendulum is again held in an upright position

Table 4.8: Cart pole setup

Parameter	Value	Parameter	Value
m_1	0.5 kg	θ	$\in [\frac{\pi}{2}, \frac{3\pi}{2}]$
m_2	0.11 kg	F_{max}	15 N
l	0.64 m	Δt	20 ms
L	1 m		

Table 4.9: Network parameters of cart pole balancing task

Parameter	Value	Parameter	Value
N	400 neurons	f_i^r	1
ρ	1	f_b^r	0
δ	1	γ	1

and that the cart is re-centered on the rail. To automate this procedure a simple energy based swing-up algorithm (Yoshida, 1999) is used to put the pendulum in the upright position after which the control is given back to the proposed IMA control algorithm allowing for the learning to continue.

For this task the input and output of the IMA controller (Figure 4.3) are defined as $\mathbf{y}(t) = [X, \theta, \dot{\theta}]$ and $x(t) = F$, respectively. The force F that can be applied onto the cart is limited to a maximum force by ensuring $|\mathbf{x}(t)| < F_{\max} = 15$ N. The network parameters for this controller are shown in Table 4.9. The introduced RLS-parameters are set to $\lambda = 1 - 10^{-6}$ and $\alpha = 1$. The initial output weights $\mathbf{w}(0)$ are normalized random values ($\mathcal{N}(0,1)$).

4.3.5.3 Results

The presented adaptive controller learns to balance the inverted pendulum within 10 minutes with an interaction time step of $\Delta t = 20$ ms. After 23 trials the pendulum is balanced within a range of $|\theta_{\max}| < 12.03^\circ$. To evaluate its performance this controller is compared to a classical model based balancing approach that uses Ackermann's formula to achieve the desired pole placement (Ackermann et al., 1993). Furthermore, the Neural Fitted Q-iteration (NFQ) algorithm (Riedmiller, 2005) was implemented so that also a non-model based algorithm is used in the comparison. The experiments for which the results are shown in Table 4.10 were conducted by Matthias Dossche during his last Master's year. Therefore, I would like to refer to Dossche (2012) for a detailed description of all the used control approaches and experiments in this comparison.

As shown in Table 4.10, the adaptive controller learns the balancing task after 10 minutes of interacting with the experimental setup. This interaction time includes the necessary time to do calculations. Compared to NFQ the interaction time is the same. However, the needed calculation time is about 4 hours. Given a faster computer and better implementation the resulting calculation time could be reduced. The standard deviation of the required force during balancing is around 12 N for the classical approach while it is 15 N for both non-model based approaches. The maximum angular deviation $|\theta_{\max}|$ during balancing is similar for the NFQ and classical approach. The angular range in which the adaptive controller succeeds in

Table 4.10: Comparison of cart pole balancing task

	Pole placement	NFQ	IMA controller
Model information	yes	no	no
Learning rule	-	RPROP	RLS
Learning mode	-	offline	online
Interaction time	-	10 min	10 min
Calculation time	-	4 h	-
σ_F	12 N	15 N	15 N
$\lvert\theta_{\max}\rvert$	10.3°	10.3°	12.03°

balancing is a little larger.

4.4 Conclusions

In this chapter I presented an novel feedback control framework, called Inverse Modeling Adaptive (IMA) control framework. The core of this framework is a dynamical system, referred to as network A or B, with a state representation is sufficiently rich (e.g. recurrent neural network) to hold an inverse model of the plant (system under control). The excitation of network B is used to generate a plant-input and eventually a plant-response. Afterwards, this pair is used to train the output weights of a network A. These weights in turn are used as the output weights of network B (weight sharing). Each iteration, network A improves its inverse model of the plant. As both networks are identical (except for their input), the controlling performance of network B, which has the desired plant-output as network input, will improve. By applying this framework, accurate control on a wide variety of plants is achieved. Learning is fast and online without the need for a pre-acquired plant model. Furthermore, the convergence of the training algorithm and a method that allows the stability analysis under which local asymptotic stability is guaranteed, were presented.

The IMA control framework was validated on several challenging control tasks with different dynamics by using Echo State Networks (ESN) as learning modules: inverse kinematics of iCub robot arm (nonlinear kinematics), the heating tank (slow nonlinear dynamics), flight pitch control (slow linear dynamics), the double inverted pendulum (fast linear and nonlinear dynamics) and the real life cart pole balancing task (fast linear and nonlinear dynamics). In most conducted experiments, the proposed IMA controller was compared to other standard control

techniques.

I have demonstrated that the adaptive controller succeeds in learning the inverse kinematics of the 7 DOF iCub arm and can generalize quite well in unseen regions of the task space.

The results of the heating tank experiments show that the IMA controller is able to react relatively quickly to changes in the desired plant-output. The tracking of different kinds of output signals (red noise signal and staircase signal) was demonstrated. Although such a varying desired output improves plant exploration, a constant desired plant output can be handled as well. The performance was compared with an existing state-of-the-art model-based control method, NEPSAC. In this comparison I have shown that the proposed IMA controller converges faster, when a moderate overshoot is allowed. The disadvantage of using NEPSAC is its slower control and, as a results, its longer convergence time. The introduced delay $\delta\Delta t$ depends on the rate at which relevant samples are presented to the network. Slow dynamics need a larger δ, fast dynamics need a smaller delay. Furthermore, I have found that in this task the improvement in overshoot by increasing δ is also limited by the memory capacity of the used network. This observation can be argued by the fact that the modeling power of the relation between $(\mathbf{y}(t-\delta\Delta t), \mathbf{y}(t)$ and $\tilde{\mathbf{x}}(t-\delta\Delta t))$ is lost when the RC-network is unable to "remember" it δ time steps later.

During the flight pitch control experiments the controller needs to switch between different desired pitch angles by controlling the elevator deflection. Due to the online learning nature of the controller, the acquired model representation of the plant improves as more samples are presented. As a result, the controller's performance increases as the experiment progresses. Furthermore, the controller's performance was compared with a classical LQR-controller. As for the previous task, the IMA controller is the most accurate if moderate convergence times are allowed.

The double inverted pendulum and cart pole balancing tasks are different from the other tasks, because with the IMA controller, which is a tracking controller, both setups can become uncontrollable outside a certain range. Due to the online learning nature of the IMA controller, this needs to be appropriately managed. By reinitializing the network states and the experimental setup, the resulting control for both task was shown to be successful. The IMA controller demonstrated that it learns the balancing of the double inverted pendulum and the cart pole within a few minutes. Furthermore, I demonstrated that, when balancing is achieved, the IMA controller is capable of balancing the pendulum, starting from a random initial position within the predefined range.

Given the presented experiments, one can conclude that the proposed IMA controller is indeed capable of learning a wide variety of control tasks. Achieving the same results with classical model-based controllers is impossible, because of

the diversity in dynamical properties. As with any model-free controller, the IMA controller's independence on prior model information, allows it to control systems for which no model is available.

5
Control Hierarchies

In previous Chapters I introduced a motion pattern generator and an adaptive control framework that together allow us to learn motion skills by demonstration, without having a model of the robot at hand. Given a set of recorded motion examples, it becomes possible to generalize to similar motions in a changing and uncertain environment. In order to explore new strategies from which robot control can benefit, researchers again turn to biological research investigating the underlying mechanisms that make humans and animals exhibit advanced yet graceful motor control.

For instance, research performed on frogs (Bizzi et al., 1991) showed that electrical microstimulation of different areas of the lumbar cord generated distinct types of force fields in the frog's isometric leg movement. Similar research on frogs (Mussa-Ivaldi et al., 1994; Kargo and Giszter, 2000) and rats (Tresch and Bizzi, 1999) has shown that simultaneous stimulation of such areas result in a superposition of the separate recorded force fields, suggesting a hierarchical/modular control system. In Mussa-Ivaldi and Bizzi (2000), this work has been extended to the planning of limb movements and how to transform this planning into a sufficient set of motor commands.

Other biological research suggests that specialized neural circuits, so called *Central Pattern Generators* (CPGs), located in the spinal cord are responsible for generating rhythmic activations needed for body function including the contractions of a heart or lungs and control of muscles for walking (Stein et al., 1999). Suggesting that some aspects of brain functions are offloaded to regions outside of the brain. These findings are based on the study of the locomotion of a lamprey, which is a primitive fish (in Ijspeert (2008) an extensive review is presented). For instance, researchers discovered (Cohen and Wallen, 1980) after extracting and isolating the spinal cord from the body, that the spinal cord, when excited with electrical stimulations, will produce fictive locomotion. This indicates that sensory information is not needed to generate such rhythmic patterns. However, it plays a crucial role in shaping the generated pattern to keep the coordination between

the CPGs and the body. Rossignol (2000) demonstrated that a mechanically driven treadmill can induce a normal looking walking gait in a decerebrated cat.

Instead of using invasive techniques to investigate the existence of a modular control system, researchers (d'Avella et al., 2003) also developed methods to find out whether a large set of natural movements is the result of a combination of a limited set of *motor primitives*, solely based on muscle activity observations. By measuring such activations with Electromyography (EMG) and applying a decomposition technique over multiple EMG recordings, they found primitive representations, called synergies. These experiments were first conducted on frogs and later on humans (Hart and Giszter, 2004; d'Avella and Bizzi, 2005; Cheung et al., 2005).

Also on the behavioral level it has been demonstrated that humans try to follow mental templates of motion when executing a task (Bernstein, 1967). The presence of these mental templates or *movement primitives* can also be detected as velocity bumps (Doeringer and Hogan, 1998) during online movement corrections. A more detailed overview of primitives at the neural, dynamic and kinematic level can be found in Flash and Hochner (2005).

Not only does such biological research help us roboticists in finding new ways to control our robots, the validation of such hierarchies on robots also allows the biologist to investigate certain hypotheses. For instance, in Schaal et al. (2003) the idea of a modular control system using primitive representations for robots was investigated. In Muelling et al. (2010) they demonstrate a robot that learned to play table tennis based on a set of primitives learned by imitating human table tennis movements. In Schaal et al. (2005), a flexible and reactive framework for motor control was presented that uses dynamic movement primitives (DMPs) (Schaal, 2006). This framework has demonstrated to be useful in the generation of walking motion of a biped based on oscillating DMPs or the generation of the swimming and walking motions of a salamander robot (Ijspeert et al., 2007) when DMPs are used as central pattern generator.

In this Chapter two different control hierarchies are presented that apply the techniques described in previous Chapters. The first Section investigates a simple hierarchy in which different levels operate on other aspects of the control system, each at different time scales. Here the MPG proposed in Chapter 3 is used in the context of a CPG generating basic motions for a robot leg. During control this CPG is modulated by a higher level controller (Chapter 4) that tries to control more slowly changing motion properties.

The second hierarchy represents a modular architecture with control primitives (MACOP) that uses a set of controllers (Chapter 4), where each controller becomes specialized in a part of the robot's state space. By enforcing a set of desired properties on the mixing mechanism, a mixture of control signals emerges unsupervised that successfully solves the control task. Here, each controller's control signal is called a control primitive.

5.1 A hierarchy of time scales for motor control

When researchers began physiological investigations of the nervous system they started with easily accessible regions such as sensory and motor elements and their combination in elementary reflexes. Based on these initial finding theories were built where reflexes were taking a central role, defining animals as reactive and reflex-driven machines. Consequently in biology motor patterns were explained as a chain of reflexes where one reflex creates an appropriate sensory stimulus that triggers another reflex. However, reflex or response chain theories were proved not to be consistent with subtle behavioral differences measured under the same conditions.

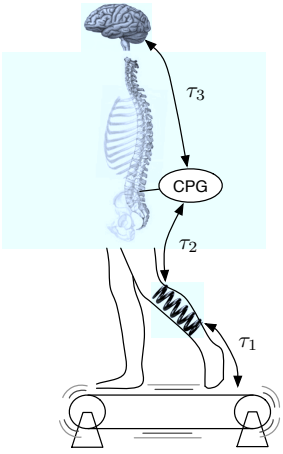


Figure 5.1: Illustration of an example hierarchical temporal structure, mapped onto a body.

Furthermore, response chains based on reflex loops cannot explain rapid movement sequences when the interval of a movement segment is smaller to the minimum latency of a response. As a result, a new theory was needed that incorporates generalized representations of behaviors that can be modulated by external and internal influences.

According to Kiebel et al. (2008), many aspects of brain functions can be explained by a hierarchy of temporal scales at which representations of the environment evolve. The higher level encodes slower contextual changes in the environment or body while at the lower level faster variations due to sensory processing are encoded.

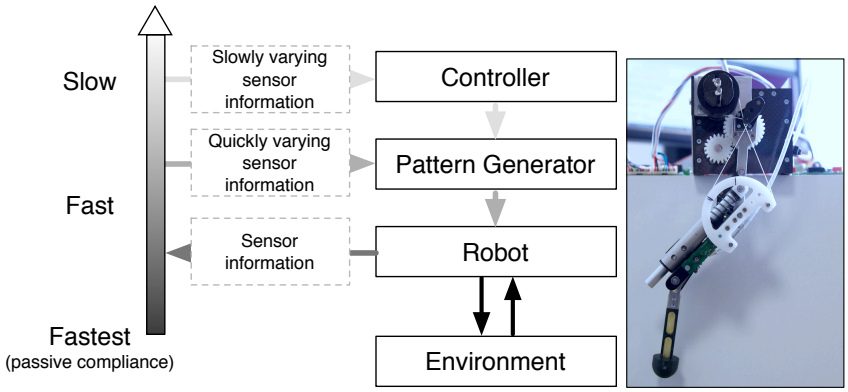
As physiological approaches for investigating the central nervous system (CNS) advanced, intrinsic motor patterns were found. These motor patterns in the CNS are produced by neural circuits even in the absence of patterned sensory inputs. Cohen and Wallen (1980), after extracting and isolating the spinal cord from a fish body, discovered that the spinal cord, when excited with electrical stimulations, will produce fictive locomotion. This indicates that sensory information is not needed to generate such periodic patterns. However, it plays a crucial role in shaping the generated pattern to keep the coordination between the CPGs and the body.

In all vertebrates, neural circuits located in the spinal cord can be found that are responsible for generating periodic or discrete activations used for locomotion

that are called Central Pattern Generators (CPGs). These CPGs (often periodic in nature) can be considered to be a specific type of motor primitive that drives a particular function of the body (e.g., breathing) (Stein et al., 1999).

Given these findings many researchers propose a both spatial and temporal hierarchal structuring of the motor control problem where a planning and commanding system is coordinating one or a set of primitives that in turn produce motor commands (Arbib, 2003). The corresponding changes in the body and the environment are registered by proprioceptive and exteroceptive sensors. Consequently, these sensor values affect every layer of the controlling hierarchy. It is interesting to investigate the spatial properties of the hierarchy where a combination of a set of primitives produces a desired motor signal. However, in this chapter, I limit the investigation to the feasibility of a single pattern generator that is being directed by a higher level system that tries to make changes at a much slower time scale based on slower sensory information than what the pattern generator is receiving. Furthermore, the morphology of the body introduces an additional lower layer that is able to react to very fast changes, much faster than all the other layers can perceive. Consider for example a human that is walking on a vibrating treadmill such as shown in Figure 5.1. The vibrations have a very small amplitude but a frequency that is too fast for the brain or CPG to handle. However, thanks to the morphological structure these small but fast (time scale τ_1) vibrations are compensated by the muscles which act as springs. As a result the CPG does not perceive these vibrations at time scale τ_2 and keeps generating the walking movement without the brain having to consciously think about it. Whenever the muscles/springs are unable to handle larger variations in the environment (e.g., sudden obstacle on the treadmill), the CPG will react by a reflex movement. As soon as the properties of the treadmill change more dramatically such as the speed or position the brain becomes conscious (time scale τ_3) about this and modulates the CPG appropriately. The brain can also become conscious about a reflex movement induced by the CPG itself, but at a slower time scale.

The proposed hierarchy, illustrated in Figure 5.2(a), consists of two building blocks: a pattern generator and a controller. On the lowest level, a pattern generator operates at a fast time scale and embeds a learned periodic pattern that is given to the motors of the robot. The rotary encoders of the motor system provide the pattern generator with direct feedback. Only environmental changes that are unhandleable by the passive compliance of the leg, will be visible for this encoder. On the highest level, and thus slower time scale, the controller tunes the parameters of the pattern generator online in such a way that it keeps track of the slow varying parameters of the resulting motor. To achieve this, the sensor information presented to the controller is preprocessed by calculating for example amplitude and offset. As a result, the proposed hierarchy operates at multiple time scales that allows the use of a more advanced controller (which often acts slower) while the



(a) (b)

Figure 5.2: (a) An overview of the control hierarchy is presented. Our approach uses two building modules working on different time scales. The first module, the pattern generator, operates at a fast time scale and gets feedback from fast varying sensors. The parameters of the pattern generator are adapted by a controller, operating at a slower time scale, that gets feedback about slowly variating motion properties. The environment is included to illustrate that very fast perturbations caused by interacting with the environment are handled by the passive compliance of the leg. (b) Shows the actual leg of the Oncilla robot build in the AMARSi consortium on which the experiments were performed.

pattern generator can be kept relatively simple and can immediately act on for instance perturbations. In other words, the motor commands generated by the pattern generator are separated from the functional control, which is done on a higher level.

5.1.1 Modulatable Motion pattern generator

As discussed previously, the robot is controlled directly by a pattern generator that is able to embed and generalize beyond demonstrated periodic and/or discrete motions. In Chapter 3 such a Motion Pattern Generator (MPG) was proposed. As shown in Figure 5.2(a) and similar to a previous experiment where the motion of a planar manipulator was generated by an MPG, the robot is also included in the feedback loop. In order to enable the controller to modulate the generated pattern a modulation approach (Section 3.3.6) needs to be chosen. For this control hierarchy, however, only the technique that changes the bias weights will be considered.

In Li and Jaeger (2011), controlling the bias weights was achieved by parameterizing $\mathbf{W}_{\text{b}}^{\text{r}}$ with a single variable for each motion property one wants to control. In this section, the control is limited to two slowly variating characteristics, namely amplitude and offset. This parametrization allows us to modify Equation (2.3) as follows:

$$\mathbf{a}(k+1) = (1-\gamma)\mathbf{a}(k) + \gamma \tanh\left(\mathbf{W}_{\text{r}}^{\text{r}}\mathbf{a}(k) + \mathbf{W}_{\text{o}}^{\text{r}}\mathbf{o}(k) + \tilde{\mathbf{W}}_{\text{b}}^{\text{r}}\right), \tag{5.1}$$

After training the MPG with $\mathbf{W}_{\text{b}}^{\text{r}}$ one can determine for each neuron how an additive bias affects the desired properties. In the case at hand $\tilde{\mathbf{W}}_{\text{b}}^{\text{r}}$ can be written as a linear combination of control vectors that represent the corresponding neurons sensitivity to a property. More formally this can be written as follows:

$$\tilde{\mathbf{W}}_{\text{b}}^{\text{r}} = \mathbf{C}_a\varepsilon_a + \mathbf{C}_o\varepsilon_o + \mathbf{W}_{\text{b}}^{\text{r}}, \tag{5.2}$$

were the amplitude and offset are represented by their corresponding parameters ε_a and ε_o, respectively. In order to prevent cross interference between desired properties the basis control vectors $\mathbf{C}_i$ are modified in such a way that all elements of $\mathbf{C}_o$ are zero where $\mathbf{C}_a$ has non zero elements, and vice versa. In other words the control vectors $\mathbf{C}_i$, are orthogonal, which means that all possible inner products are $\mathbf{0}$. For the amplitude and offset this can be written more formally as:

$$\langle \mathbf{C}_a, \mathbf{C}_o \rangle = \mathbf{0}. \tag{5.3}$$

Their exist several methods to find the control vectors $\mathbf{C}_i$. In Li and Jaeger (2011) and Jaeger (2010) two different approaches were presented, one based on correlations and one on perturbations. More recently a simpler and more intuitive approach was proposed by my colleague Francis wyffels in wyffels (2013). Without going into much detail, it relies on determining the difference between the current neuron states and a moving average of these neuron states, while the system is driven by a signal that changes monotonically in a certain property (e.g., amplitude or offset). For instance, if the trained MPG is driven (via its feedback) by an example motion that increases in amplitude one can see which neurons' state changes quickly compared to the other and previous neuron states. These neurons are therefore more sensitive to amplitude changes and in turn affect the amplitude more than others. Changing ε_a and ε_o will therefore modulate the amplitude and offset, respectively.

When controlling a real robot the timing at which new positions are fed to the robot is important, especially when smooth and reactive motion behavior is required. One of the benefits of using the proposed MPG is that it is computationally fast to produce a new position. Therefore, the rate at which the robot can be controlled is not limited by the MPG but rather by the hardware itself.

5.1.2 Controller

In order to learn how the parameters ε_a and ε_o should change in Equation 5.2 to allow for the desired motion properties to be achieved, the IMA controller proposed in Chapter 4 is used. The plant in this control task is the above described MPG attached to the robot, which means that the controller is driving the MPG and observes the properties of the resulting leg motion. If the measured amplitude and offset are denoted by $\xi_a(t)$ and $\xi_o(t)$, respectively, the controller input is defined as $\mathbf{y}(t) = [\xi_a(t), \xi_o(t)]^T$. In contrast, the controlled parameter are $\mathbf{x}(t) = [\varepsilon_a, \varepsilon_o]^T$. The target values for both the amplitude and offset are given as desired plant-output.

The IMA controller uses an ESN to create an inverse model of the task. Depending on this ESN's network size, it is possible that the calculation time for a new control command is slower than the hardware limitations to the control rate. The computational burden of this learning algorithm and thus the control rate of the controller depends on the size of the network used. For a simple task (e.g., linear low dimensional task) a controller with a smaller network might be sufficient while having the advantage of being fast. A larger network however might be slower but can model much more complex tasks.

An advantage of the controller used is that it learns to control the task at hand while interacting with it given all the task depending restrictions, including the given control rate. However, this does not mean that the controller will succeed when for instance trying to balance a pendulum by sending new commands every 10 seconds.

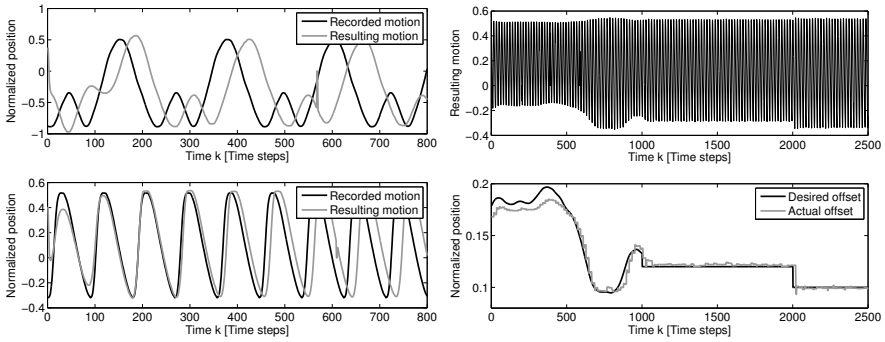
5.1.3 Experiments

In this section the discussed control hierarchy is applied on a prototype robot leg of the Oncilla robot platform, which is developed in the AMARSi consortium and shown in Figure 5.2(b). This robot leg is controlled by a motor control board that is driven by a small computer. However, because of the computational limitations of this onboard computer and to ensure the desired communication timings, all calculations are offloaded to a much more suited computational unit.

The purpose of these experiments is to evaluate the described control hierarchy concept on a simple task. The number of experiments is quite limited and should be considered to be a proof of concept.

Although the control of multiple servos is possible, the experiments are simplified so that only one servo is used. This servo is controlled by a simple P-controller, which converts the positions, generated by the MPG, to a torque signal. However, to allow for changes in the robot dynamics to be visible in its motion, the used P-parameter is smaller than optimal and the amount of torque is limited.

Table 5.1 gives an overview of the MPG and IMA controller parameters used during the experiments. Additionally, the different timings are shown at which



(a) (b)

Figure 5.3: (a) Shows two different recorded motions together with the actual reproduction by the robot leg. (b) Depicts the actual motion during offset control (top) together with the desired and actual offset (bottom).

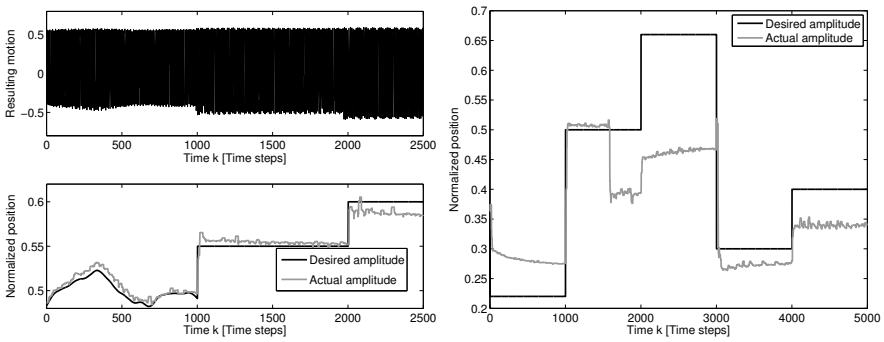
each system is interacting with another layer of the hierarchy. As mentioned in the previous section, the IMA controller is interacting at a much slower rate compared to the MPG's control rate.

Table 5.1: Summary of all our setup parameters used in the experiments.

Parameter	Pattern generator	Controller
N	500	500
ρ	1.4	1.
γ	0.14	1.
α	0.1	0.01
f_i^r	1.0	0.1
f_b^r	0.5	0.5
Δt	20ms	100ms

5.1.3.1 Learning by imitation

By limiting the servo's torque the robot leg can be back-driven, allowing the demonstration of a desired motion. In this experiment a periodic motion is imposed which afterwards is used to train the MPG-network. When training is completed, the



(a) (b)

Figure 5.4: (a) Depicts the actual motion during amplitude control (top) together with the desired and actual amplitude (bottom). (b) This plot illustrates how the proposed hierarchy reacts to changes in the dynamics of the robot leg during amplitude control.

necessary gradient vectors $\mathbf{C}_i$ to the MPG-neurons that affect the amplitude and offset, are computed. In Figure 5.3(a) the actual trained leg motion is shown for two different imposed patterns that are shown as well. The first pattern is a periodic pattern similar as in a swing/stance phase gait while the second motion is a sinusoidal pattern. Both resulting motions show a phase shift caused by integrating the robot leg into the feedback loop of the MPG. A change in the generated pattern has to propagate through the dynamics of the robot leg, before the correct leg angle is visible for the leg encoder. Because of the feedback loop, this variable delay affects the generated pattern. This illustrates that a higher level control is necessary to modulate the pattern generator in such a way that these dynamics are taken into account.

5.1.3.2 Motion modulation by controlling the MPG

After learning the recorded motion, the feedback controller is applied to modulate the amplitude and offset of the motion, which are only observable on a slower time scale. As mentioned before in Section 5.1.1, this can be achieved by controlling ε_a and ε_o in Equation 5.2. Figure 5.3(b) shows the desired and actual offset, which are controlled by the feedback controller. To control the offset, the highest level of the proposed hierarchy was interacting with the MPG every 100 ms (5 times slower than the interaction rate of the MPG). Figure 5.4(a) demonstrates a similar experiment but for amplitude control. Additionally, the actual resulting positions are depicted at the bottom of both Figure 5.3(b) and 5.4(a). When observing both

the results for the offset control and the amplitude control, one can notice that controlling the offset affects the amplitude and vice versa. In order to prevent this undesired behavior, the degrees-of-freedom of the control task need to be restricted. As one can imagine, their might exist more than one solution to control a single motion property. Therefore, controlling both the amplitude and offset is necessary to get the desired motion response.

5.1.3.3 Adapting to changes in robot dynamics

In the previous experiment it was shown that the generated motion can be modulated. However, one needs to investigate the capability of the proposed hierarchy to adapt to changes in the dynamics of the robot or in its environment. Although the purpose of this single prototype leg is not to interact with an environment yet, it is possible to impose a change in dynamics by increasing the mass of the robot leg. In order to achieve such change an extra weight is added to the tip of the leg. As a result, the amplitude of the motion will be reduced and the offset will move closer to the lowest point of the leg. However, this switch in dynamics will cause the inverse model of the feedback controller to adapt to these changes as well. As a result, during amplitude control the amplitude should eventually converge again to its desired value. In Fig 5.4(b) after 1500 time steps a mass of 100 g is added to the leg. After adjusting its internal model, the controller starts compensating for the extra weight at time step 3000 by controlling the bias of the MPG. In retrospect, the target signal of the amplitude was changing too fast to clearly observe final convergence. One can clearly observe, during the first 1000 time steps, that the actual produced amplitude is still converging to the desired amplitude when the target is changed to 0.5. After adding the extra weight, the target should be kept constant for a longer period of time, in order to demonstrate the model adaptation more clearly.

5.1.4 Conclusions

In this section, I described a multi-timescale control hierarchy, inspired by physiological research, that uses random dynamical systems for each layer. On the lowest level, a motion pattern generator (MPG) is able to embed any periodic signal. This pattern generator interacts directly with the motor of the leg on a fast timescale. On a higher level, and thus slower time scale, an IMA controller tunes the parameters of the pattern generator online in order to keep track of the slowly varying parameters of the resulting motion. To achieve this, the sensor information presented to the controller is preprocessed by calculating for example the amplitude and offset. Since the controller acts on a slower timescale, this controller can be very advanced and might consist of a very large random dynamical system. On the other hand, the

pattern generator is fast enough to react immediately on small perturbations that cannot be compensated by the morphology of the robot (passive compliance).

By means of three simple experiments on the AMARSi Oncilla leg, the proposed control hierarchy is validated. In a first experiment I showed that the hierarchy is able to capture a (by hand) shown periodic motion pattern that is embedded by the pattern generator. In the second experiment I demonstrated the ability of the higher level controller to track slow varying properties such as amplitude and offset, by only controlling the bias of the pattern generator. Finally, in the third experiment, it is investigated that the control hierarchy is able to deal with new situations such as changes of the leg weight. In retrospect, the target signal changed too fast to convincingly demonstrate the adaptability of the IMA controller. However, in Section 5.2.7.1 the IMA controller's ability to adapt to new situations will be demonstrated more clearly, in such a way that any possible doubt is eliminated.

Instead of having a single pattern generator it should be possible to define a set of pattern generators, each generating a unique pattern, that can be combined by the controller to generate a more skillful robot movement (Appendix A.2). When this is possible, the multi-timescale hierarchy, addresses the investigation of a robot movement generation in a bottom-up manner, from a predefined set of motion primitives to the final robot movement. Another approach would be to take a top-down approach where the motion primitives emerge unsupervised based on a high-level description of how primitives should generally behave. In the following section, such an approach is investigated and evaluated on a robotic tasks.

5.2 MACOP: Modular Architecture with Control Primitives

In Chapter 4 the IMA control framework was introduced and evaluated on several control tasks. One of the tasks used for this evaluation was to learn the inverse kinematics of the iCub robot arm. The attained results demonstrate that the IMA controller is able to fulfill the following:

- Learning an implicit mapping from actions to the resulting trajectory without prior knowledge.
- Because of a fixed control rate, the dynamic behavior of the robot is anticipated.
- Handling redundancies because controlling multiple joints results in a mapping which is not single-valued.

The range in which the iCub's arm can move is, however, quite limited compared to some industrial robots like the Programmable Universal Machine for Assembly

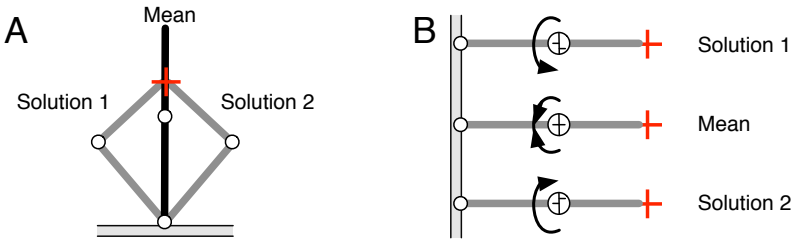


Figure 5.5: Panel **A** illustrates the averaging effect of a 2D robot arm when two different training solutions (gray) are used to train an ESN. The black arm represents the resulting pose when the joint angles are averaged out. As a result, the desired end-effector position (red cross) is not reached. Panel **B** demonstrates a situation where the mean of the joint angles results in an arm position that still reaches the desired end-effector position.

(PUMA) arm. The actual *redundancies*[1] present in the inverse kinematics (IK) learning task are therefore limited as well. After trying to extend the IK learning application to the PUMA robot, where each joint can move in a larger range, I found out that the dramatic increase in possible poses for the same end-effector position was causing the control framework to fail in learning the IK. As discussed in Chapter 4, the initial noisy control commands are used as exploration, which can lead to redundant training data. Training for instance an ESN with such training data causes the approach to average out between these redundant solutions, yielding an inaccurate control (shown in Figure 5.5). However, for some rotational joint configurations such averaging behavior does not affect the accuracy.

In order to solve this redundancy problem one could use a learning approach that can generate multiple solutions to the same problem. However, the architecture proposed in this Chapter uses a different approach. Instead of having a single controller with multiple solutions as output, the architecture uses multiple controllers over which redundant solutions can be distributed. Although the proposed architecture, called MACOP, is applicable on a wide variety of tasks, I will explain the design of MACOP by means of an inverse kinematic learning task.

[1] When the number of degrees of freedom is large enough, there will exist multiple solutions for the same task. One of these solution can be kept while the others are redundant and removable without affecting the task.

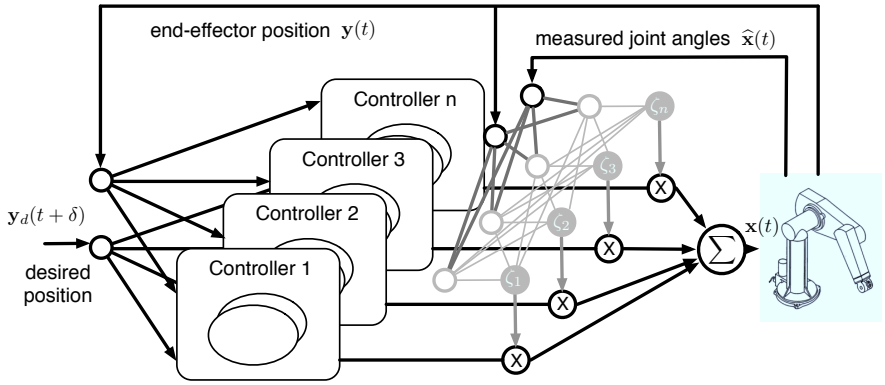


Figure 5.6: Illustration of MACOP, which consists of an ensemble of IMA controllers depicted in Figure 4.3. The desired (objective) and the current end-effector position are used as external inputs to each controller. The controller outputs are weighted by a scaling factor ζ_i and superimposed with each other in such a way that the resulting joint angles control the robot. The used ζ_i represents the responsibility of a controller and is determined by the measured joint angles and end-effector position.

5.2.1 PUMA Robot Arm Platform

In this work all the experiments are performed on a dynamic Webots simulation model of the PUMA 500 robot arm that has 6 DOF (degrees of freedom). This numerical model allows the architecture to apply joint angles and measure its actual values in the dynamical environment of the robot. Every $\Delta t = 32$ ms (default Webots configuration) the architecture interacts with the simulation model, which means that between every sample taken, 32 ms passed in the simulation environment. The controlled joint angles of the robot arm are converted to joint torques by PID controllers. Each joint is equipped with an encoder to measure the actual joint angles. Additionally, the simulation environment provides the Euclidean end-effector position of the robot arm.

5.2.2 Design of MACOP

As mentioned above, controlling a system such as a robot arm with an inverse modeling approach, like the one described in Chapter 4, poses some difficulties when multiple solutions are possible. However, often the modeling complexity of a control problem can be reduced by decomposing the problem into less complex

parts. For instance, if we consider a cup lifting scenario for which we want to learn a model. This model will approximate the underlying dynamics of the particular task. If we extend this problem to lifting other objects, the resulting model needs to be able to approximate a single function that includes both tasks with different dynamics. Solving such problem requires very complex and hard to achieve models.

One of the approaches that can solve such problems is called MOSAIC (Modular Selection and Identification for Control) (Wolpert and Kawato, 1998; Haruno et al., 2001), which suggest a feasible strategy on how a human motor control system learns and adapts to new dynamic characteristics of the environment. MOSAIC learns a different controller for each task, and uses a 'responsibility'-function to decide which controller will be used, while still allowing for smooth switching between the controllers individual actions. When introducing a new task, MOSAIC will generalize, by combining the actions of each controller. To determine each controller's 'responsibility', every controller contains a forward model that predicts the next state of the object based on the previous control actions. If a controller's forward model is more accurate compared to the others, that controller's inverse model is trained further with the new observations and used to control the robot arm.

One potential weakness of this approach is that the performance of a forward model is not necessarily a good indicator of the modeling performance of the inverse model. To confirm this, I have tried an approach directly based on MOSAIC. For each IMA controller, an ESN (see Chapter 2) was trained to serve as both an inverse and forward model at the same time. Together with my colleague Michiel Hermans, we found that all forward models had initially roughly the same prediction error, leading to an equal responsibility factor for each controller as a result. During the training phase, however, small variations in performance error influenced both the training of the forward and the inverse models. Eventually this always causes one controller to be fully responsible at all times, making the other controllers redundant. These findings confirm the observations made by Haruno et al. (2001) even though in the original MOSAIC setup, the inverse and forward models are completely separate from each other, meaning that there is no relation at all between the modeling performance of the inverse and forward model. Based on these experiments one can argue that the responsibility of a controller is fully determined by noise on the forward modeling performance of a controller. Any other controller selection mechanism might thus be as useful as the one used by MOSAIC.

Therefore the Modular Architecture for Control with Primitives (MACOP) is proposed, which is inspired on MOSAIC, and depicted in Figure 5.6. Instead of using both an inverse and forward model, only an inverse model is used to produce actions for the robot arm. Determining when an IMA controller should contribute

to the task is learned unsupervised given the robot's state and some desired mixing properties. This can be related to a Kohonen map (Kohonen, 1998).

5.2.3 Controller Selection

As depicted in Figure 5.6, the actual control signal is a weighted sum of the outputs of a limited number of controller outputs. The weight factors, which are the equivalent of MOSAIC's 'responsibility', depend on observable properties of the robot. Each controller learns to control the robot arm by creating an inverse robot model. Simultaneously, the mixing mechanism is trained online. In order for a controller to distinguish itself from others, the rate at which each controller is trained will be modulated according to its corresponding responsibility. This will be explained in more detail when I describe the operation of a single controller.

Suppose we have N_c controllers. The output of the i-th controller is denoted as $\mathbf{x}_i(t)$. The controlled joint angles $\mathbf{x}(t)$ are then given by:

$$\mathbf{x}(t) = \sum_{i=1}^{N_c} \zeta_i(t)\mathbf{x}_i(t), \tag{5.4}$$

where $\zeta_i(t)$ is the scaling factor, or 'responsibility' of a controller, that determines how much each controller is expressed in the final control signal. Ideally, one would like to let $\zeta_i(t)$ express the momentary accuracy of each controller. For example, if each controller is randomly initialized before training, certain controllers may be better than others when the arm is near a certain pose, and one would like to use the ζ_i to scale up the control signal of these controllers, and suppress that of the others. In reality, however, the accuracy of each individual controller cannot be directly measured because the robot is driven with the weighted sum of the control signals, and not the individual controllers.

Therefore, a different strategy will be applied. I will introduce a way in which the scaling factors will automatically start to represent local parts of the operating regime of the robot, and next the associated controllers will specialize to be more accurate within this local area.

We wish $\zeta_i(t)$ to only depend on the current end-effector position $\mathbf{y}(t)$ and the measured joint angles $\widehat{\mathbf{x}}(t)$, both of which are observable properties of the robot arm. As each controller will attempt to learn an inverse model, the combined control signal will need to be of the same magnitude as the individual control signals. Therefore, we will make sure that the scaling factors are always positive, and always sum to one:

$$\sum_{i=1}^{N_c} \zeta_i(t) = 1 \qquad \text{and} \qquad 0 \leq \zeta_i(t) \leq 1. \tag{5.5}$$

Both these qualities can be ensured if ζ_i is calculated by a *softmax* function. First

a linear projection from the joint angles and end effector position to a vector $\mathbf{r}$ is used:

$$\mathbf{r}(t) = \begin{bmatrix} r_1(t) \\ r_2(t) \\ \vdots \\ r_{N_c}(t) \end{bmatrix} = \mathbf{V}(t) \begin{bmatrix} \mathbf{y}(t) \\ \widehat{\mathbf{x}}(t) \end{bmatrix}. \tag{5.6}$$

Next, the softmax function is computed.

$$\zeta_i(t) = \frac{\exp(r_i(t))}{\sum_{j=1}^{N_c} \exp(r_j(t))}. \tag{5.7}$$

The projection matrix $\mathbf{V}(t)$ is a matrix of size N_c by $N_\mathbf{y} + N_{\widehat{\mathbf{x}}}$, where $N_\mathbf{y}$ and $N_{\widehat{\mathbf{x}}}$ are the dimensions of $\mathbf{y}(t)$ and $\widehat{\mathbf{x}}(t)$, respectively. Dependence on time comes from the fact that, like all parameters, $\mathbf{V}$ is trained online.

$\mathbf{V}(t)$ is randomly initialised, with elements drawn from a normal distribution $\mathcal{N}(0, 0.1)$. It will determine how the responsibilities are distributed. $\mathbf{V}(t)$ needs to be trained in such a way that MACOP learns to generate the target trajectories by mixing all contributions as desired. In this work I wish to obtain the following two qualitative properties:

- Each controller should contribute in a unique way to the movement generation of the robot. In order for a controller to distinguish itself from the others, its corresponding $\zeta_i(t)$ should peak over the others. It is desired that there is sufficient variation, so that at each moment in time some controllers are significantly more responsible than others.
- On the other hand, it should be prevented that there are no responsibilities that are close to zero at all times, such that all controllers are put to good use, and the potential power of the ensemble is fully exploited. We wish to avoid the observed situation when implementing the MOSAIC-based controller ensemble, where eventually only one controller contributed to the task.

Based on these two desired properties, a learning algorithm is constructed for training $\mathbf{V}(t)$. The first property of our mixing mechanism can be achieved by gradually increasing the magnitude of $\mathbf{V}$. This results in a more strongly peaked distribution for the scaling factors. This can be understood by looking at the limit situations. If all elements of $\mathbf{V}(t)$ are zero, all scaling factors are equal. If the magnitude goes to infinity, the softmax function will be equal to one for the highest element, and zero for all others. Controlling the magnitude of $\mathbf{V}(t)$ allows us to make a smooth transition between these extremes. It was chosen to increase the magnitude of $\mathbf{V}(t)$ linearly each time step by adding a small increment, equal to $\mathbf{V}(t)$ normalized with its Frobenius norm. As a result, the small increment is independent of the actual magnitude of $\mathbf{V}(t)$.

The second mixing property requires that all controllers contribute significantly to the robot motion. The manner in which I chose to do this was to suppress the scaling of the momentary maximal scaling factor. This ensures that no single scaling factor can remain dominant for a long time. Suppressing one scaling factor automatically scales up the others, allowing that in the end none of the scaling factors remains very small at all times. In order to train $\mathbf{V}(t)$ to get this effect, target values for the $\zeta_i(t)$ need to be set at each time step. The target value of the highest $\zeta_i(t)$ is set equal to N_c^{-1} (which would be the long-term time average of all scaling factors if they all contribute equally). At the same time we have to make sure that the sum of the target values is equal to one (i.e., a target that can be reached by a softmax function). To obtain this, the target values of the other $\zeta_i(t)$ are equal to themselves, scaled up to ensure that the sum of the targets equals one. If target value for $\zeta_i(t)$ is denoted as $\theta_i(t)$, one can write

$$\theta_i(t) = \begin{cases} h(t)\zeta_i(t), & \text{if } i \neq \text{argmax}(\zeta_i(t)) \\ \frac{1}{N_c} & \text{if } i = \text{argmax}(\zeta_i(t)) \end{cases}, \tag{5.8}$$

with

$$h(t) = \frac{1 - N_c^{-1}}{1 - \max(\zeta_i(t))}. \tag{5.9}$$

To train $\mathbf{V}$ according to these target values, the gradient of the cross-entropy[2] $H(\theta_i, \zeta_i)$ is calculated with respect to $\mathbf{V}$. For both desired properties an update rule is defined and each time step both contributions are added, resulting in the following update rule:

$$\mathbf{V}(t+\Delta t) = \mathbf{V}(t) + \eta_g \frac{\mathbf{V}(t)}{||\mathbf{V}(t)||_F} + \eta_s [\mathbf{y}^\mathsf{T}(t), \widehat{\mathbf{x}}^\mathsf{T}(t)](\boldsymbol{\zeta}(t) - \boldsymbol{\theta}(t)), \tag{5.10}$$

where $\boldsymbol{\zeta}(t)$ and $\boldsymbol{\theta}(t)$ are column vectors with the responsibility factors and their targets, respectively, and η_g and η_s are two learning rates. In order to prevent one mixing property to dominate the other, these learning rates are set such that both properties are present.

In order to visualize the segmentation by the projection matrix $\mathbf{V}$, I have applied it on a simple rotational actuator. In this experiment I used 4 controllers. The actuator can be controlled by applying a desired angular velocity $\mathbf{x} = [\dot{\theta}]$. The actual angular position $\mathbf{y} = [\theta]$ and velocity $\widehat{\mathbf{x}}$ are measured and used to calculate Equation 5.6. In Figure 5.7, the results are shown for 3 different tasks in which

[2]Here the scaling factors and their target values are treated as if they were probabilities, which stems from the common use of a softmax function: to model a multinomial distribution function (Bishop et al., 2006) is calculated. One could as well use mean-square error to train $\mathbf{V}(t)$ on the target values, but the resulting update equations would be more complicated, whereas cross-entropy leads to a simple formula.

the actuator is controlled to move periodically between $\pm\pi$, $\pm\frac{\pi}{2}$ or $\pm\frac{\pi}{4}$, respectively. The actuator and the corresponding range for each task are illustrated at the bottom of Figure 5.7. The second row visualizes the state space of the actuator, where the state trajectory of the periodic motion is depicted by a black circle. For every possible state space position the dominant controller is determined, which allows us to color the entire state space accordingly. Based on this coloring one can clearly notice the wedge shape segmentation of the two dimensional state space. All the controllers and the initialization of the projection matrix $\mathbf{V}$, are the same for all three experiments. Therefore, the difference in segmentation for every task, is purely caused by the task it self. These changes to $\mathbf{V}$, can be better visualized by showing how much every controller is contributing, instead of only coloring according to the dominant controller. Now consider that every plot in the middle row is described by a polar reference frame. Because of the wedge shaped segmentation, the coloring only changes for certain angles, but not for any radius. So, given a certain angle, the dimension of the radius can be used to show how much all controllers are contributing. Now, consider a segmented line of which each of the 4 segments has a different color and the length of each segment corresponds to the value of ζ_i. By using the dimension of the radius to show this segmented line, for a given angle, we get the disks at the top of Figure 5.7 if we do this for all possible angles. Depending on the desired range of the periodic signal, the color pattern clearly changes in size and rotation. Later in this Chapter, it will become clear that the controller's performance depends on the used segmentation. However, this little experiment also illustrates that the segmentation is affected by the control. There exists thus a bidirectional dependency between the segmentation and the control.

When the learning rates are fixed, Equation (5.10) never converges. What will happen is that the magnitude of $\mathbf{V}$ slowly keeps on increasing, and in the long run, the scaling factor distribution will become highly peaked[3]. Therefore, during all our experiments, unless mentioned differently, the root mean-square-error (RMSE) between the desired and the measured end-effector position[4] is calculated over a moving time-window of 1000 samples. When this RMSE becomes smaller than 1 mm, both η_g and η_s are linearly decreased over the course of 5000 samples until they reach 0. After this point the elements of $\mathbf{V}$ no longer change.

5.2.4 Single Controller

Because only inverse models are used for controlling the multi-jointed robot arm the IMA controllers described in Chapter 4 will be used.

[3] At each moment, one will be close to one, the others close to zero

[4] This is the average distance, so that one can express RMSE in millimeter or centimeter.

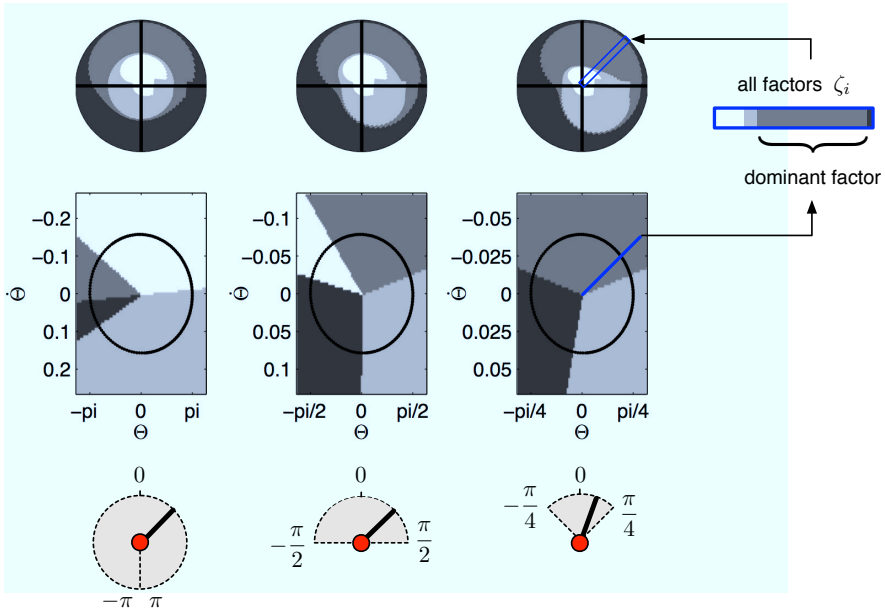


Figure 5.7: Illustration of how the projection matrix $\mathbf{V}$ is segmenting the state space. The bottom row depicts the three different tasks where an actuator is moving over three different ranges. The middle row depicts the resulting state space with the corresponding state space trajectory of the motion. The coloring is done according to the dominance scaling factor. The top row, however, depicts all scaling fractions on a disk.

5.2.4.1 General setup

For the inverse kinematic learning task of MACOP the end-effector position (outcome) is denoted by $\mathbf{y}(t)$, and the joint angles (the actions) by $\mathbf{x}(t)$. It is assumed that a model (model A in Figure 4.3) can be trained to approximate the past joint angles $\mathbf{x}(t-\delta\Delta t)$, $\delta\Delta t$ being a fixed delay period, given that it receives the current and the delayed end-effector position $\mathbf{y}(t)$ and $\mathbf{y}(t-\delta\Delta t)$, respectively.

In general, the optimal delay δ depends on the rate at which the dynamics are observed Δt (sample rate) and the kind of dynamics that are inherent to the control task. Plants with fast dynamics usually require a smaller δ than slower dynamical systems when using the same sample rate. For this task $\delta = 1$, because one time step delay is sufficient to capture the dynamics of the task at hand. More details concerning this parameter can be found in Chapter 4.

In order to improve exploration and to speed up training in the initial training phase, a small amount of noise is added (initially drawn from $\mathcal{N}(0,7)$ in mm) to the desired end-effector position y_d, of which the standard deviation linearly diminishes to 0 over the course of $50,000$ samples of training.

In principle, there is no need to ever stop training the inverse model in the IMA controller. Indeed, if the experimenter knows that the conditions of the setup may change over time, it could be desirable to keep the online learning mechanism active at all times in order to let it keep track of changes in the system. In the performed experiments, however, I chose to gradually slow down the learning algorithm and at some point in time let it stop, in order that all parameters in the controller architecture remain fixed during testing. More details are provided in Section 5.2.4.4.

5.2.4.2 Echo State Networks

The IMA controller in Chapter 4 can use any dynamical system with a high dimensional state representation to create an inverse model. However, ESNs will be used for all our experiments unless mentioned differently. Furthermore, during our experiments the input and training signal to the ESN is scaled ensuring that their values are between -1 and 1. Consequently, this scaling needs to be undone before the generated network output represents actual joint angles that can control the robot. The ESN parameters that are common for all our experiments are shown in Table 5.2.

5.2.4.3 Linear Controllers

In order to check how much the operation of MACOP depends on the type of controller, I also conducted an experiment in which linear controllers are used. Here, the output of the inverse model is a direct linear combination of its input, so no

Table 5.2: Network parameters for the inverse kinematics learning task

Parameter	Value	Parameter	Value
ρ	1	f_b^r	0.1
δ	1	γ	1
f_i^r	0.1		

ESN and thus no non-linearity or memory is present in these controllers. As a result, learning the non-linear part of the full inverse kinematics will largely need to be accounted for by training the responsibility factors. Here too, the system will be trained online, according to the algorithm described in section 5.2.4.4.

5.2.4.4 Training

In order to train the IMA controllers online, the Recursive Least Squares algorithm described in Section 2.1.5.4 is used. With each iteration the output weights $\mathbf{W_r^o}(t)$ are adjusted in order that the network converges to the desired output. However, the rate at which these weights are changed is controlled by the corresponding responsibility factor ζ_i. Within the proposed MACOP architecture, such adaptive learning rate allows each IMA controller's inverse model to distinguish itself from the other controllers. Additionally, in order to allow the weights to converge to fixed values, the training speed is modulated with a factor $l(t)$. Therefore, the only equation from the RLS algorithm that needs to be adjusted is Equation (2.19). This weight update equation can thus be adjusted as follows[5]:

$$\mathbf{W_r^o}(t) = \mathbf{W_r^o}(t-\Delta t) - l(t)\zeta(t)\mathbf{e}(t)(\mathbf{P}(t)\mathbf{a}(t))^{\mathsf{T}}. \tag{5.11}$$

The error $\mathbf{e}(t)$ is the difference between the actual and the desired joint angles. To allow $\mathbf{W_r^o}(t)$ to converge together with the projection matrix $\mathbf{V}$ from Equation (5.10), $l(t)$ is decreased linearly from 1 to 0 in the same fashion as the learning rates η_g and η_s in Equation (5.10), i.e., as long as the average error over the last 1000 time steps is larger than 1 mm, it is equal to one, and as soon as it is smaller, it linearly decreases to zero over the course of 5000 time steps.

[5]Note that I use k in Equation (2.19) to indicate the time step. Here, however t indicates the actual time: $t = k\Delta t$.

5.2.5 Analyzing MACOP

Each IMA controller learns to produce a set of control commands, which in this task, contributes to solving the IK problem. These control commands produced by a controller are called a control primitive. To analyze MACOP's behavior and each controller's contribution, several approaches are useful:

- **Tracking a trajectory:** As described above, MACOP is designed in such a way that the scaling of a controller depends on the location of the actual tracked state (e.g., end-effector position) and the actual actuator state (e.g., robot's pose when commanding joint angles). A control primitive that has the largest 'responsibility' (biggest $\zeta_i(t)$), is called the dominant primitive and is generated by the dominant controller. Given a desired trajectory it is interesting to compare the time course of the scaling factors, and how they relate to the actual tracked state of the robot. The trajectory of the tracked state is colored according to which IMA controller is the dominant one at that state, this in order to show which controllers specialize in which regions of task space.

- **Selecting a single controller:** Even when the scaling factors strongly fluctuate in time, this does not necessarily mean that the controllers are sending different control signals. Indeed, even though the learning speed is modulated according to the scaling factors, all of them still are trained to perform the same task. In order to verify if specialization indeed occurs, experiments are conducted in which after the training phase, only one IMA controller is used (i.e., its scaling factor is set to 1 and all others to 0). Given the resulting trajectories the individual model performance can be compared to its true scaling factor.

- **One versus multiple controllers:** One of the main research questions of this section is of course how much can one profit from using MACOP versus a single controller. In order to answer this, the performance of the setup is measured as a function of the number of controllers. To allow a fair comparison, the number of trainable parameters (the total number of output weights of the ESN) remains constant for each setup.

5.2.6 Results

All training parameters for the experiment are provided in Table 5.3. The RLS parameters λ and α are chosen based on previous experience. The learning rates η_g and η_s are found by trial and error, but it was experimentally verified that performance does not change much in a broad region around the provided values.

Table 5.3: Simulation parameters

Parameter	Value	Parameter	Value
λ	$1-10^{-4}$	η_g	0.00008
α	0.01	η_s	0.0002

5.2.6.1 Tracking a trajectory

To investigate the overall behavior of MACOP, I applied several desired robot end-effector trajectories. Both Figure 5.8 and 5.9 show the resulting trajectories (after convergence) of following a rectangular and circular-shaped target trajectory. In both experiments an RMSE of 10 cm was achieved within $10,000$ samples and a RMSE of 1 mm (point of convergence) within $100,000$ samples, demonstrating that the system is able to follow a desired trajectory closely.

In the first experiment, MACOP is trained to generate a rectangular trajectory that spirals back and forth into the X-direction over several passes. For this task 5 IMA controllers are used, each with 50 neurons. The top panel of Figure 5.8 shows the trajectory generated by the robot after convergence (all learning rates are 0 and RMSE = 1 mm). Each part of the trajectory is colored according to which controller is dominant at that time. The responsibilities $\zeta_i(t)$ themselves are shown in the bottom panel of Figure 5.8. It appears that the 4 controllers have formed a specialization for certain regions of task space, their responsibilities ζ_i peaking at the corresponding parts of the trajectory.

Notice that the depth of the trajectory in the X-direction is rather small (20 cm), and yet the scaling factors strongly vary as a function of it. This is especially apparent in the green and blue parts of the trajectory. This strong change in scaling factor is caused mostly by the pose of the robot, and not so much the end-effector position, as I have verified by experimentally testing the sensitivity of the scaling factors as a function of the joint angles and position. This suggest that the control architecture effectively uses information of the robot pose to solve the task.

A second experiment ($N_c = 4$, 50 neurons each) extends the difficulty of the previous trajectory to demonstrate responsibility/task space correlations over a larger time period of the desired movement. The trajectory describes four passes of a circle in a single direction during 8 seconds, after which the trajectory smoothly switches to describing a shifted circle in the opposite direction of the previous circle.

The results after convergence are shown in Figure 5.9. The rotation direction of the trajectory is indicated by the arrows. In the left part of the circular trajectory the blue controller is significantly contributing to the control of the robot. This contribution is reduced when the robot is performing the right circular movement.

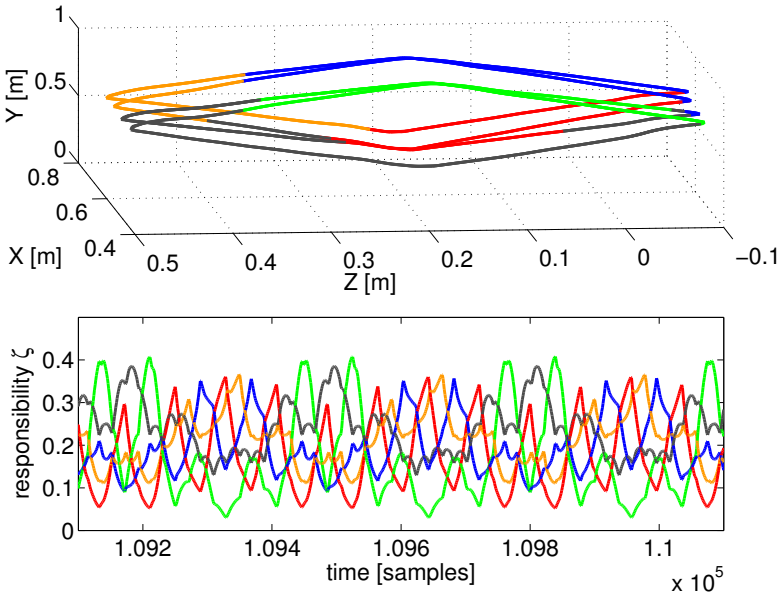


Figure 5.8: Top panel: the resulting end-effector trajectory generated by the robot arm for the rectangular target trajectory after convergence (RMSE < 1 mm for the full trajectory). The corresponding color of the dominant controller is shown. Bottom panel: the responsibility factors $\zeta_i(t)$ as a function of time.

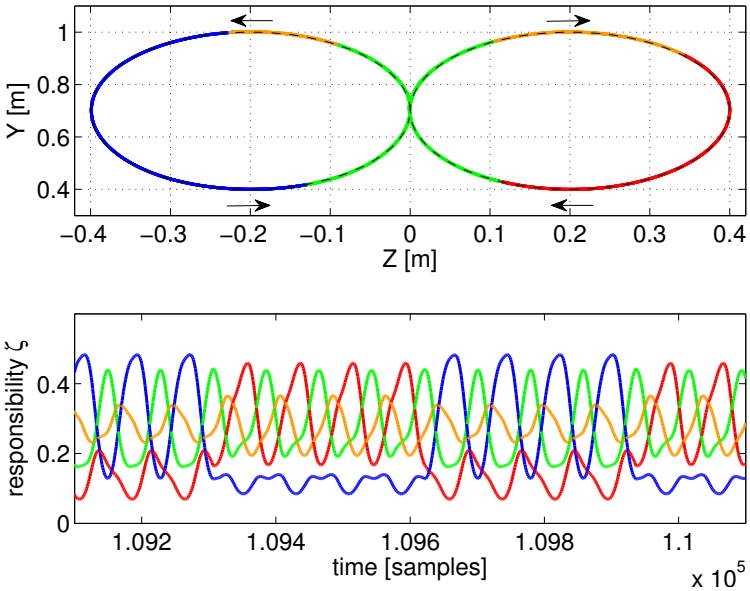


Figure 5.9: Top panel: the resulting end-effector trajectory generated by the robot arm for the circular target trajectory after convergence. The corresponding color of the dominant controller is shown. The arrows indicate the direction of the trajectory. Bottom panel: the responsibility factors $\zeta_i(t)$ as a function of time.

In this part of the trajectory, the red controller is contributing more.

In order to verify MACOPs robustness, the double circle trajectory after training is considered, of which a part of the trajectory is eliminated so that there exists a discontinuous jump in the target end-effector position. The result is shown in Figure 5.10. It appears that after a large initial overshoot, the robot recovers and is eventually capable of tracking the desired trajectory again. The overshoot can be largely explained by the fact that the controller never saw a discontinuous jump during training. Consequently, it has seen no examples of what happens when large torques are applied on the joints. Furthermore, the sudden jump forces the robot arm into a region of task space where it never resided during training, causing unpredictable behavior. In robotic applications, such behavior is undesirable. In order to prevent this, one can use an additional velocity or torque control mechanism, or allow for the controllers to adapt to such changes.

After training MACOP (same configuration as before) with the double circle trajectory a test grid is defined on the plane of the training trajectory with a resolution of 1 cm. The test target points of this trajectory are visited by sweeping the grid in both directions of the Z-axis. The result of such an experiment is shown in the last two rows of Figure 5.10. Each pixel represents the RMSE (in meters) of a specific grid point. Averaged over 10 experiments (different initialization and training) a mean RMSE $= 4.4$ mm with a standard deviation of 3.1 mm is achieved. Note that the RMSE in the grid corners are bigger because they are harder to reach. When moving from right to left, the arm configuration from one point to another can be different for the opposite direction. Hence, the difference in both plots.

In the final experiment of this section, a set of linear controllers are used to see if MACOP is able to still control the robot arm to generate a trajectory with very low-complexity controllers. I found that at least 9 controllers are needed to approximate the target trajectory with a final RMSE $= 1.5$ cm. Using MACOP with fewer controllers does not work. Figure 5.11 shows the resulting trajectory and the scaling factors of the individual controllers. The fact that MACOP is capable of solving the tracking task with such basic controllers is a strong indicator that the presented training algorithm for the scaling factors is quite successful in distributing the complexity of the full task. It also demonstrates that MACOP can be easily extended to include any kind of inverse model.

5.2.6.2 One versus multiple controllers

One of the main assumptions underlying MACOP is that it is beneficial to distribute the full control task over multiple controllers. I first investigated if the mixing is in fact responsible for the increase in performance, and not just having several distinct controllers in the first place. This was examined by keeping the responsibility factor $\zeta_i(t)$ constant and equal to N_c^{-1} for each model, and this in the case of 5 IMA controllers. It turns out that this situation leads to the same performance

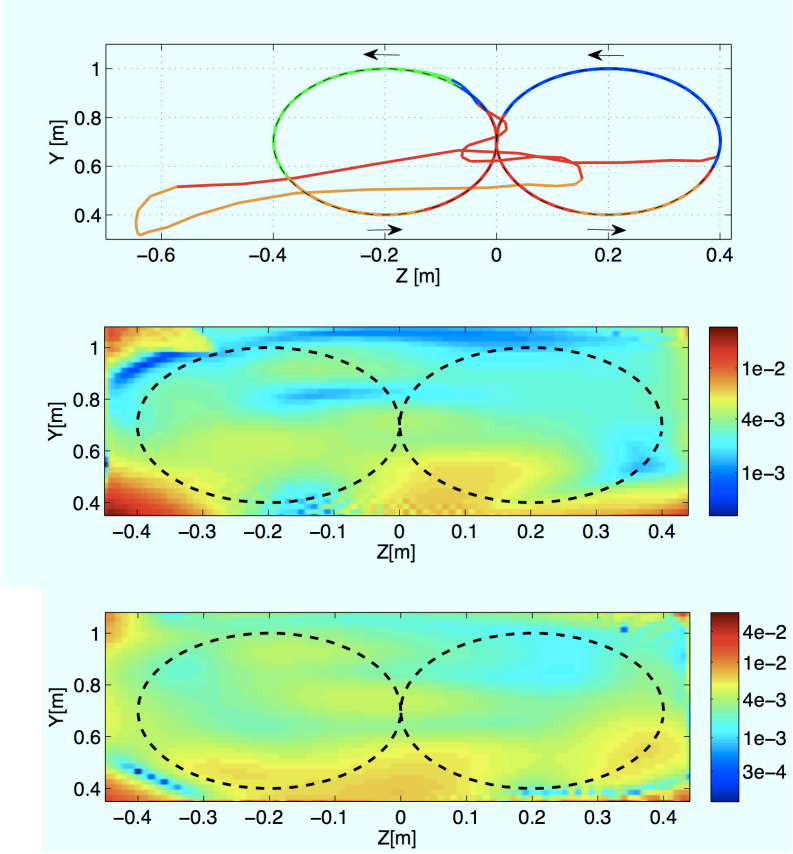


Figure 5.10: Top panel: a part of the generated trajectory before the switch and after a jump, where the dominant controllers are represented by a corresponding color. The direction in which the trajectory is tracked is indicated by the arrows. Two bottom rows: visualization of the generalization performance of the learned IK (circular training trajectory: dashed line) on a test grid. The bottom plot represents the results when sweeping from left to right, while the plot above is visualizing the sweep from right to left. Color scale is the RMSE in m.

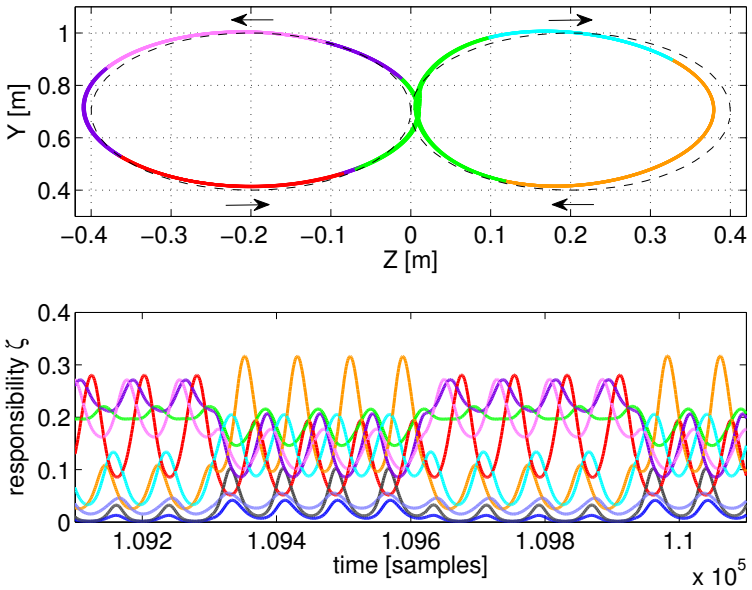


Figure 5.11: Illustration of the tracking performance of MACOP with 9 linear controllers after convergence. Top panel: the resulting trajectory, colored according to the dominant controller. The dashed line is the target. Bottom panel: the corresponding responsibilities as a function of time.

that is attained by using a single, large reservoir (which performs worse, as I will show next), showing that the variable responsibility factors directly increases performance.

To investigate how much the tracking performance depends on the number of controllers an experiment was conducted in which the mean error between the produced and target trajectory was measured for different number of controllers. The distance is measured from the end-effector to the target averaged over 5000 samples after training. All learning rates are linearly reduced to 0 after $100,000$ samples of training (because some experiments will never reach the requirement of an RMSE less than 1 mm). For an increasing number of controllers MACOP is applied on trajectories which are based on all 26 letters in the English alphabet, which I have drawn by hand. As the trajectory is repeated periodically, I also made sure that the end and starting point are the same in each trajectory (to avoid sudden jumps). After recording the trajectories they are scaled and placed in the YZ-plane.

In each experiment a randomly initialized controller ensemble is trained to produce a single letter of which the RMSE is measured after convergence. For each number of controllers, the average RMSE over 50 instantiations of each letter is determined, so that the measured result for each N_c is averaged over $50 \times 26 = 1300$ experiments. In order to keep the comparison fair, the number of trainable parameters (the total number of output weights of all the ESNs) remain constant. In practice this means that I used 250 neurons for a single controller, 125 for 2 controllers, etc. In Figure 5.12 the results are presented. A single controller performs rather poorly. The optimal number of controllers for the entire English alphabet is around 6 controllers. When the number of controllers increases further, the number of neurons, and hence the modeling power, of each controller becomes smaller. Similar to the experiment with the linear controller, this experiment shows that a great deal of the modeling complexity is covered by the mixing mechanism.

It should be noted that in some cases, due to the random initialization, the robot can get stuck in a certain pose (as some joint angles are limited between certain values), and never reaches the desired trajectory. This is the reason why the maximum values in Figure 5.12 are much bigger than the mean over all experiments. If these cases are disregarded, a RMSE of 1 mm is measured within $100,000$ training samples.

5.2.6.3 Control Primitives

A single controller's contribution is called a control primitive and differs from the regular notion of primitives. In this section the properties of such control primitives are investigated in order to evaluate this definition of a control primitive. Due to the MACOP setup it is not straightforward to get a good understanding of the role of a single controller. At all points in time, all controllers influence the robot, and due to the feedback, each controller influences all other controllers. One can think of

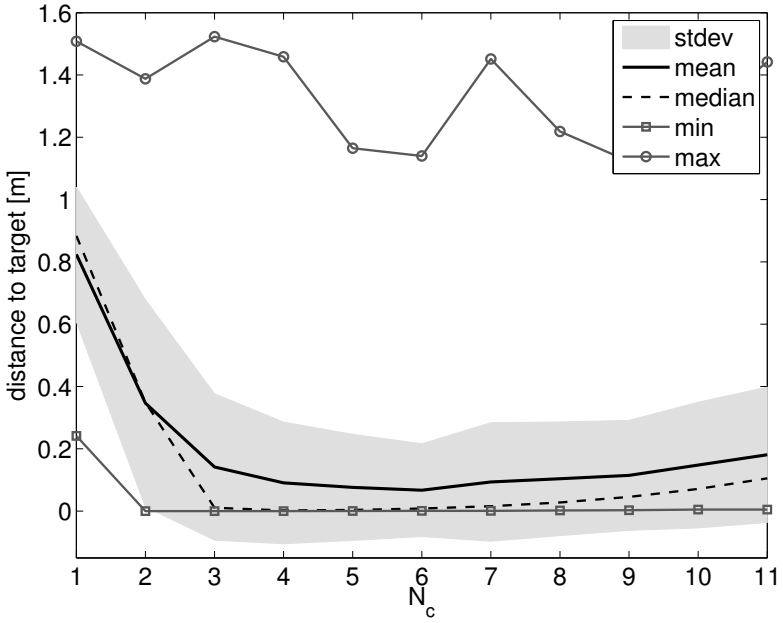


Figure 5.12: Effect of the number of controllers on the tracking performance of MACOP on the English alphabet trajectories. The mean, median, standard deviation, minimum and maximum values over 1300 experiments are shown for each controller configuration.

two ways in which to study the individual controller contributions. Either we use a single controller in the ensemble (with scaling set to one) for steering the robot, which then ignores potential feedback by the influence of the other controllers, and as such emergent synergies are not expressed. Alternatively, we could record the control signals of the individual controllers during normal operation, and use these recordings to steer the robot afterwards. Even though this approach will take into account potential synergies between the controllers, during testing it has no feedback at all, allowing for the trajectory to start drifting from the objective. In this setup, however, it seems that such an effect does not occur. Therefore, this approach is used. I have tried the other approach as well, and the results were qualitatively similar.

To get a qualitative idea of how the individual contributions look, I revisit the task inspired by the English alphabet, and train a controller ensemble to draw the letters of the word 'amarsi'[6] one after another. After training, the recorded contributions of a single unscaled controller is used to steer the Webots simulation, and the robot's response is recorded.

The result is shown in Figure 5.13(a). The five rows starting from the top show what trajectory each individual control primitive produces if it alone is present in the control architecture (in a corresponding colour), plotted over the target value (grey). The bottom row shows the trajectory of the full ensemble, colored according to the most dominant controller. It is interesting to note that, even though all individual controllers produce a trajectory that resembles the target, all of them strongly deviate from the true target, and each of them produces a distinctly different response. The scaled combination of them, however, tracks the objective far more closely, which again indicates that the mixing mechanism works well to combine contributions of several controllers.

A second experiment that is performed is to see whether true specialization occurs. After all, even though one controller is dominant, the other controllers will also contribute to the total motion. In order to check this, I have conducted a similar experiment to the double-circle objective, depicted in Figure 5.9. Here, four IMA controllers are used, and the individual recordings are used to drive the robot. Next, the distance error of the end effector as a function of time is recorded which is then compared with the corresponding scaling factor of the controller. If specialization occurs, we would expect to see some negative correlation between the error on the trajectory and the corresponding scaling of the controller. If the scaling factor of a certain controller is high, this would indicate that it specializes in the current region of task space, and the resulting error should be low. Vice versa, if the scaling is low the controller should perform worse, as it is not its region of specialization.

The result for each controller is shown in Figure 5.13(b). For some controllers there seems to be a strong relation between the error and the scaling of the corre-

[6] AMARSi is an EU project concerning adaptive motor skills for robots.

sponding model. Especially in the case of the red and blue controller. The relation is weaker, however, for the other two. Indeed, the scaling factors for these two controllers fluctuate less, in order that they are able to train their corresponding inverse models throughout the full trajectory, leading to better overall generalization. From this it can be concluded that MACOP uses both specialization and signal mixing to obtain good control over the robot arm.

Motor and motion primitives generally refer to different building blocks at different levels of the motor hierarchy. They can be kinematic (e.g., strokes, sub movements), dynamic (e.g., joint torque synergies, control policies) or both. According to Flash and Hochner (2005) their crucial feature is that a wide variety of movements can be derived from a limited number of stored primitives by applying appropriate transformations. Within this definition a controller's contribution to the joint angles can be called a primitive. Their organization is stored within the mixing transformation so that after convergence a consistent controller selection is achieved. What is different from the common interpretation of primitives is that in our case, the control primitives are mixed and rescaled constantly, instead of truly being selected and weighted statically.

MACOP learns to spread a set of controllers in the vicinity of the target trajectory in such a way that primitives produced by controllers can help in tracking this trajectory. Even when the complexity of these controller is reduced (using linear controllers), the task is still solvable. Unlike in Nori and Frezza (2004), the MACOP control primitives and their mixing values both depend on the state of the robot, and because they are adapted online they become dependent on the task. After convergence, however, this task dependency is removed.

5.2.6.4 Coping with dynamic effects

An inverse kinematic mapping maps a desired task space position to the corresponding joint angles. In most evaluations of learning inverse kinematics, a new command is only sent when the previous desired joint angles are reached. For this, a direct inverse mapping without memory suffices. However, in all conducted experiments MACOP does not wait for the robot to reach the target joint angles and send new target joint angles at a constant rate (every 32 ms). As shown in Figure 5.14, a PID-controller, which applies the necessary torque to reach a desired joint angle, has a dynamic transition before reaching a new target angle. These dynamic transitions need to be compensated by the IMA controllers as well, in order to reach the desired outcome in time[7]. Such transitions require memory instead of a direct mapping. MACOP's ability to cope with these dynamic effects is evaluated by changing the P-parameter of each joint's PID-controller (which determines how

[7] This means that the control signal the robot receives can no longer be simply interpreted as joint angles, but more as a type of motor commands.

fast the robot can react to changes in the desired joint angles) in order to increase the importance of such dynamic effects. To demonstrate this ability, MACOP is applied to the robot with the square objective that was used in Figure 5.8, but reduce the velocity of each joint. The *P*-parameter in the PID-controllers is reduced from 10 to 2, of which the effect is shown in Figure 5.14. Furthermore, in order to assess the effect of a sudden jump, the square is periodically shifted by half a meter, which introduces discontinuities in the objective.

Figure 5.15 shows the Z-coordinate as a function of time for the desired trajectory and resulting trajectories for the different *P*-parameters. The top panel depicts the results during the beginning of the experiment (30.4 s to 38.4 s), while the bottom panel depicts the results after convergence. The robot with the standard velocity (blue line) is able to follow the objective more closely during the beginning of the experiment but exhibits some fluctuations. The Z-position trajectory of the reduced velocity configuration is unable to reach the objective closely during the first part an clearly needs more time to learn the inverse model, indicating that the control problem is harder when the robot reacts more slowly. If we look at the result after convergence, we notice that the small fluctuations in the blue trajectory are reduced, and that MACOP has learned to follow the objective closely. Interestingly, due to a larger maximum velocity, the blue trajectory has a large overshoot when the applied objective exhibits a sudden jump. The red trajectory exhibits almost no overshoot in the beginning of the experiment but after convergence, the red curve also exhibits some overshoot. In the beginning of the experiment it is clear that the limited velocity of the joints causes the robot's end-effector to not reach its target in time, as opposed to the version with fast control. This indicates that the desired trajectory is more difficult to obtain if the robot has slow dynamics. Indeed, the sudden changes in direction can be made far more easily if the robot has a fast control response.

Reducing the velocity of each robot joint has its advantages in terms of power consumption and safety. However, a trade-off must be made between faster convergence, as in closely following the objective, and the amount of overshoot allowed. In some tasks it might be possible that the objective changes very fast, and in such cases, reducing the reaction speed will restrict the robot in reaching its targets.

5.2.7 Other applications

5.2.7.1 Whole Arm Manipulator

Above I described the design of MACOP by means of an inverse kinematic learning example on a simulation model of the PUMA robot. Although simulations are useful for the development of control algorithms, the performance and applicability of such algorithms should be evaluated on real robots under realistic conditions. Therefore, I visited the Learning Algorithms and Systems Laboratory (LASA) at

Table 5.4: Network parameters for WAM application

Parameter	Value	Parameter	Value
N	100	f_i^r	0.3
ρ	1	f_b^r	0.1
δ	8	γ	1

EPFL in Switzerland led by Aude Billard to apply MACOP on the 7DOF Whole Arm Manipulator (WAM) from Barrett Technology. Before going into more detail about the actual control of the robot and the corresponding experiments, I would like to discuss some practical aspects of this problem.

When using MACOP on a real robot, it is no longer possible for the robot to freeze all dynamics while MACOP is processing the robot's state and calculating new joint angles. Although the MACOP algorithm is generally fast because most calculations are simple matrix multiplications, the online learning rule (RLS) slows down the algorithm dramatically. The larger the networks the slower it becomes. However, when trying to model the WAM's inverse kinematics the networks should not become too small. The internal controllers of the WAM require a fixed control rate of 200 Hz, which means that a new command should be sent every 5 ms. Given the MACOP and network parameters, shown in Table 5.4, a fixed control rate of 25 Hz could be achieved. During the time that MACOP is busy and the WAM controller is requesting a new command, the previous MACOP commands are used.

Another difference with the PUMA robots is that the WAM is controlled by torques instead of joint angles. Often, the necessary torques to reach a desired position are determined by PID controllers. The control gain of such PID controller can become really large when the robots motion is perturbed. As a result, the corresponding torque can become dangerous. Fortunately, the WAM is equipped with a safety system in which all control signals are ignored. Generally the equations

Table 5.5: Network parameters for WAM application

		θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7
Normal	K_P	900	2500	600	500	50	50	8
	K_D	10	20	5	2	0.5	0.5	0.05
Adjusted	K_P	600	1666	400	333	33	33	5
	K_D	10	20	5	2	0.5	0.5	0.05

of motion of a robot mechanism can be described by the joint space formulation of the canonical form:

$$\mathcal{H}(\theta)\ddot{\theta} + \mathcal{C}(\theta,\dot{\theta})\dot{\theta} + \tau_g(\theta) = \tau, \tag{5.12}$$

where $\mathcal{H}(\theta)\ddot{\theta}$ and $\mathcal{C}(\theta,\dot{\theta})\dot{\theta}$ represent the joint space inertia matrix and the Coriolis/centrifugal force related matrix, respectively. The joint angles of the robot are represented by θ. The gravity related torques are denoted by $\tau_g(\theta)$, while the actual control torque is denoted by τ. $\tau_g(\theta)$ only depends on the pose of the robot, not its velocity. During all experiments only this gravity compensating term was calculated based on a rigid body model of the WAM. The other terms were accounted for by a PD controller, which yields the following control torque:

$$\tau = \tau_g(\theta) + K_P(\theta_{\mathbf{d}} - \theta) - K_D\dot{\theta}. \tag{5.13}$$

Here, K_P and K_D denote the PD control parameters which are shown together with adjusted control parameters (related to the last experiment in this section) in Table 5.5. The desired joint position provided by MACOP is denoted by $\theta_{\mathbf{d}}$. Given the MACOP configuration illustrated in Figure 5.6, $\mathbf{y}(t)$ represents the end-effector position and $\mathbf{x} = \theta_{\mathbf{d}}$ the joint angles.

The IMA controllers from Chapter 4 initially uses random control signals (motor babbling) in order to approximate an inverse model of the robot. Applying such control signals on a real robot is possible if the motor babbling is restricted to safe robot configurations where the movements have low velocities. Another approach that was used with the WAM, is to do an initial exploration on a simulation model of the robot. This model should not necessarily be accurate because the online learning allows the algorithm to adjust itself on the real robot after pre-training it on an inaccurate model. Researchers from LASA created, however, a quit accurate simulator that I used during the initial exploration phase. When a successful trajectory tracking was achieved on the simulator, MACOP's state was saved and applied to the real robot. Because of the inaccuracies of the simulation model, compared to the real robot, MACOP needs to adapt its IMA controllers so that a better tracking performance is achieved. After conducting some initial tests with the WAM, I noticed that the WAM is sensitive to small fluctuations in the control signals. Therefore, MACOP's control signals are first given to a critical damped filter before they drive the robot. As a result the tracking velocity of the trajectory is much more smooth which also yields a δ (the number of time steps the observations are shifted in MACOP) that is much larger than with the PUMA experiments.

Unlike the experiments conducted on the PUMA robot, all learning rates were kept constant during the entire duration of each experiment. Consequently, this means that no convergence of the learning was enforced. The reason for this is that the experiments on the WAM were conducted with a previous version of MACOP where no convergence was considered. Next, I will describe some basic experi-

ments I have conducted during my visit at LASA.

TRACKING A TRAJECTORY As a first evaluation of MACOP on the WAM I used some examples recorded on the actual robot. Together with Mohammad Khansari-Zadeh (post-doc at LASA) I used a Vicon motion capture system to track a marker placed on my fist (shown in Figure 5.16(a)). The position of this marker was recorded as desired trajectory that needs be tracked with MACOP. Additionally, the position of my fist was applied on the actual WAM during the recording itself because the trajectory should be recorded in a frame of reference close to the robot, instead of close to the person who is demonstrating the movement. I started with the recording of a simple trajectory that contains 3 consecutive passes of a circular movement. Afterwards, this recording was extended to a longer trajectory by concatenating the recording with itself. With this target trajectory MACOP is applied on the WAM after pre-training it on a simulation model. During this experiment 4 controllers were used each having 100 neurons (all parameters shown in Table 5.4). As shown in the top panel of Figure 5.16(b), the initial end-effector trajectory of the WAM largely differs from the target trajectory which is a single pass of the demonstrated example. However, as time progresses, demonstrated at different time instances, the tracking becomes more and more accurate. The corresponding responsibilities are shown in the bottom panel of Figure 5.16(b). The scaling factors change during the entire experiment because no convergence was enforced. After 5 trials[8] the average tracking distance of such trial is 1 cm.

In order to evaluate MACOP on a more complex trajectory I recorded, the writing of the word "amarsi", similar to the PUMA experiment that was discussed previously. Because of the added complexity, 5 IMA controllers were used for this task. After recording a single pass of the word "amarsi", where each letter is drawn over the other, the dataset was also extended by concatenating it multiple times with itself. The results shown in Figure 5.17(a), represents 3 trials on the WAM after pre-training MACOP on the simulator. The transition between each letter is not shown. As time progresses, shown from left to right and top to bottom, the tracking performance of MACOP clearly improves. If we look at the first letter written with the WAM, one can notice that that other controllers are dominant than during the remainder of the experiment. Furthermore, if we compare the blue and red part of the letter R with that of the letter S, we notice a different coloring for the same regions because each letter is drawn in the same region of the task space[9]. This clearly indicates that the controller selection not only depends on the task space location but also the pose of the WAM (joint space).

[8] One trial corresponds to the tracking of the entire recorded demonstration, without concatenation.

[9] Instead of writing the word amarsi in one fluent motion each letter next to the other, I returned to the same region as the previous letter before writing a new one.

COPING WITH DYNAMIC EFFECTS The last set of experiments I conducted on the WAM demonstrate the ability of MACOP to cope with dynamic effects, similar to the corresponding PUMA experiments. As shown in Equation (5.13), the applied torque τ consists of a PD controller and a term that accounts for the gravitational forces of a static rigid-body model of the robot. By adding or removing weight to the robot, that is not compensated by $\tau_g(\theta)$ it becomes interesting to see how MACOP copes with such changes. However, the used PD controller is able to compensate for such changes as well, without the need of MACOP to adapt its control. Therefore, I reduced the PD controller's K_P parameter for each joint as shown in Table 5.5. The only way now to compensate for the change in the WAM's weight is to generate different joint angles by adjusting its inverse kinematic model that result in the tracking of a target trajectory. During these experiments I used the circular recorded motion from previous experiments as target and applied MACOP with 4 controllers.

To change the weight of the robot I used different end-effector mounts and earmuffs that could be added to the robot's end-effector during an experiment. As shown in Figure 5.16(a), each mount has a different weight. The first mount is the one I used in all previous experiments. The second mount includes a force sensor and haptic ball, which adds around 300 g to the end-effector. Furthermore, the different geometry affects the center of gravity of the entire robot. Finally, the used earmuffs weigh around 300 g which enables us to add a maximum additional weight of around 600 g to the robots weight. In order to evaluate the tracking performance for different configurations the average tracking error during a trial is measured, and this for several trials. As shown in Figure 5.17(b) the trajectory tracking under the standard configuration (no added weight) achieves a average tracking error of 1 cm after 5 passes/trials of the recorded trajectory. This experiment was stopped after 5 trials although more trials would have resulted in an even smaller tracking error. For the second configuration the standard mount was replaced by the haptic ball and the tracking performance was measured during 14 trials. The results, shown in Figure 5.17(b), demonstrate that after the first trial the error reduces drastically after which it slowly decreases during the subsequent trials. I repeated this experiment but added the earmuffs during the second trial. The corresponding curve clearly indicates the effect of the introduced weight change. However, after 14 trials MACOP has almost compensated for the added weight. The sudden error bump during the third trial is caused by the global (all joints angles) changes MACOP is using, to compensate for the added weight. Please note that the variance in initial error measured during the first trial is caused by the initial learning phase of MACOP where the large weight changes of each controller's output weights differs from one experiment to another.

5.2.8 Conclusions

In this section I described a modular architecture with control primitives (MACOP), which learns to control a robot arm based on a pre-set number of controllers. The inspiration for this architecture stems from MOSAIC (Haruno et al., 2001), which is a control framework, inspired by a plausible model on how human motor control learns and adapts to novel environments. MOSAIC uses a strategy in which an ensemble of inverse-model controllers is trained, one for each environment with different properties. On top of this, a selection mechanism selects which controller needs to be active at which moment in time. Each controller is associated with a forward model of the system that needs to be controlled, and controller selection happens by choosing the forward model that is the most accurate.

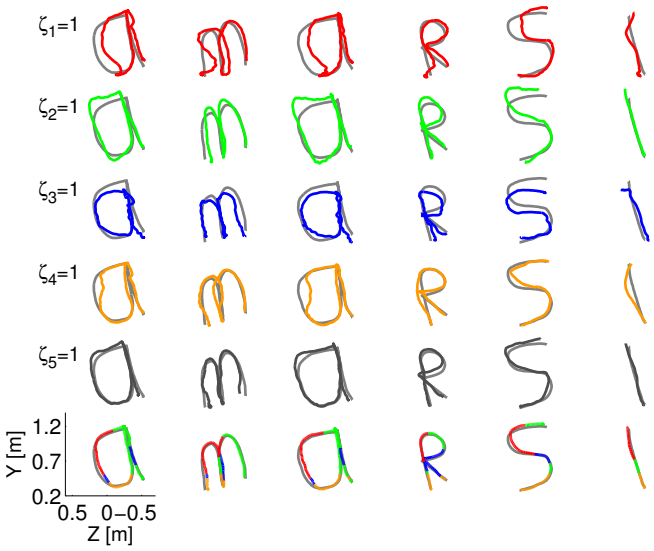
Together with my colleague Michiel Hermans we (and others (Haruno et al., 2001)) made the observations that such a strategy may not be optimal. There is no reason why the accuracy of a forward model should be correlated to that of the inverse model. Another selection mechanism of the controllers might thus be possible. In this section, I wanted to build upon the idea of a fixed number of control primitives that are continuously combined to produce a desired motion. Each Inverse Modeling Adaptive (IMA) controller used in MACOP consists of an inverse model, which is trained online and consists of an Echo State Network. Given some high-level control requirements, an unsupervised division of the task and joint space is achieved, which can be related to a Kohonen map (Kohonen, 1998). By means of an inverse kinematic learning task, I show that the mixing mechanism learns a subdivision of the entire task and joint space and produces one scaling factor for each controller that are associated with the current end-effector and joint angle position of the robot. The training error of each controller is scaled with the same factor in order that training data within a controller's associated part of the subdivision becomes more important than other data. As a result of this data selection mechanism, every controller can specialize its function within its appointed part of the joint and task space. The used mixing requirements prescribe that all controllers should contribute significantly to the task, while still allowing for a controller to specialize itself for a certain subregion.

I validated MACOP on an inverse kinematic learning task where I controlled both a simulation model (PUMA, 6DOF) and a real robot (WAM, 7DOF) by producing joint angles, which are sent at a fixed control rate. This is in contrast with other approaches such as Oyama et al. (2001) where a static mapping from task space position to joint space is learned and where a separate feedback control loop to approach the target joint angles is needed. Such a separate feedback control system results in high control gains when there is an external perturbation of the robot's movement. Achieving a compliant kinematic control thus argues for a dynamic learning approach that learns the control at a fixed control rate. MACOP therefore relies on the approach proposed in Chapter 4 for such a dynamic con-

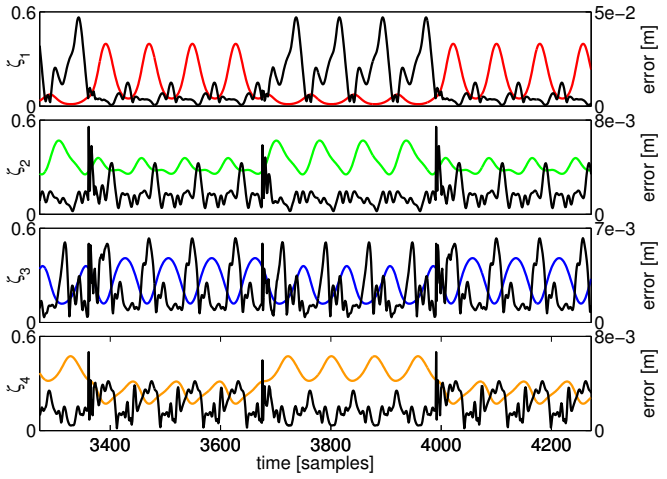
trol method. As a result MACOP is well suited to cope with the dynamical effects introduced by the non-instantaneous control of the robot: even when the robot responds slowly to the control signal, the MACOP architecture is able to compensate for it and produce the target trajectory. Furthermore, it was demonstrated on the WAM that adding weight to the robot, which is not compensated by the gravity compensating control torque, can be counteracted by MACOP. By adjusting its internal model, a different set of joint angles, corresponding to a certain end-effector position, is generated that neutralizes the additional weight.

Each controller was also replaced with a simple linear controller to validate MACOP's independence of the chosen IMA controller. Such a linear controller is constructed by learning a linear combination of the architecture's input. When the number of linear controllers within MACOP is large enough, the end-effector will start to track the target. However, the tracking performance of MACOP with the IMA controllers is better due to its non-linear nature.

The control signals produced by a single controller within MACOP are called a control primitive. Generally, however, motor and motion primitives refer to building blocks used at multiple levels of the motor hierarchy and can be dynamic and/or kinematic in nature. According to Flash and Hochner (2005), a primitive is a building block that is combined, by an appropriate transformation, with a limited number of other primitives to generate a wide variety of movements. Given this definition a controller's contribution can indeed be called a primitive. After convergence of all learning rates within MACOP, the controllers organization is stored within the mixing transformation, achieving a consistent controller selection. The control primitives are mixed and rescaled constantly, which is different from the common interpretation of primitives where they are truly being selected and weighted statically.



(a)



(b)

Figure 5.13: (a) Overview of the resulting trajectory with the "amarsi" target by using a single controller contribution. Each row shows the resulting end-effector trajectory of the robot arm. From left to right a part of the continuous writing is shown ensuring that every letter of the word "amarsi" is presented. The coloring of the trajectory illustrates which controllers contribution is used (one for each row). The bottom row show the target trajectory, together with the actual generated trajectory. (b) The error of a single controller contribution (the black curves), plotted with their corresponding scaling factors (colored) as a function of time. The vertical scale of the error is provided on the left vertical axis, and that of the scaling on the right.

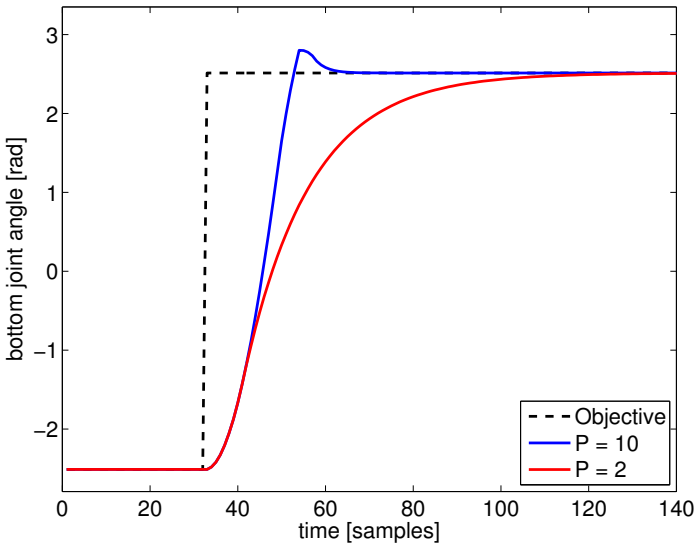


Figure 5.14: Illustration of a sudden change in desired joint angle of the bottom joint (black dashed line) of the PUMA 500. The robot response is shown for this particular joint with different *P*-parameters, to illustrate the effect of a changed *P* value. By decreasing the *P*-parameter the dynamic transition time from one position to the other increases.

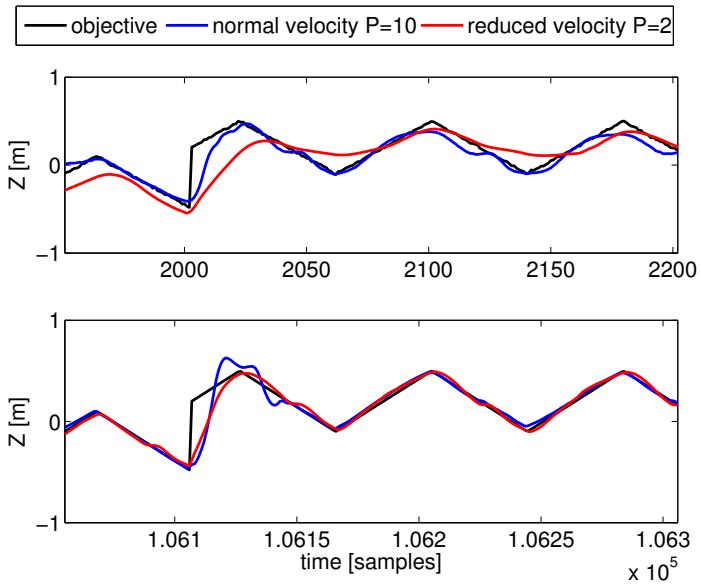
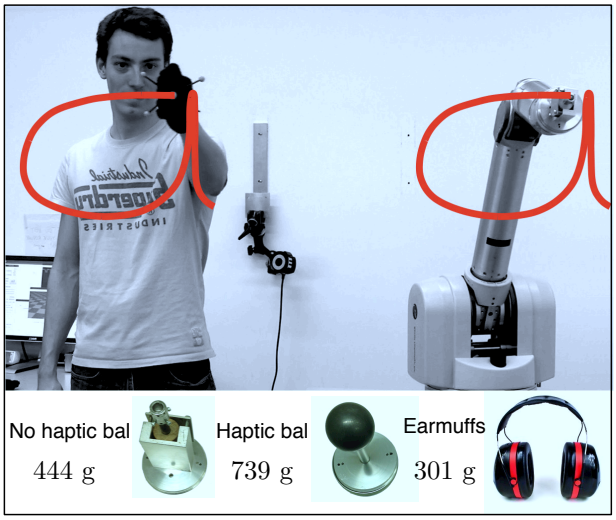
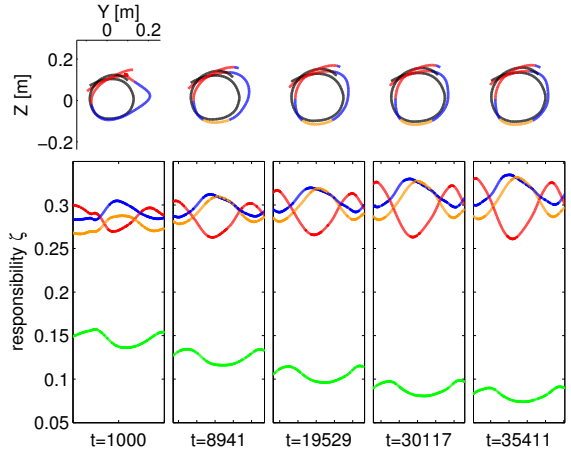


Figure 5.15: Comparison of the Z-coordinate of the generated trajectories for the shifted-squares objective. Shown are the desired Z-coordinate (black), and those generated by robots with $P = 10$ (blue), and $P = 2$ (red). The top panel is during the early parts of the training, and the bottom one after convergence. The abrupt change in the desired trajectory corresponds to the shift of the squares.

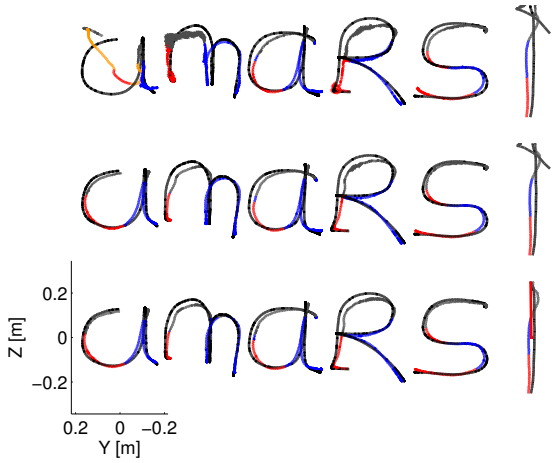


(a)

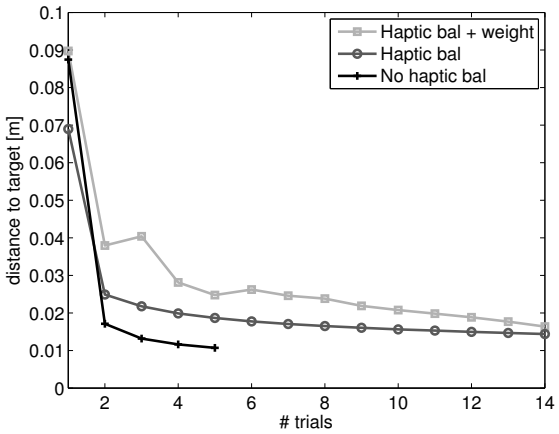


(b)

Figure 5.16: (a) This image illustrates the motion capture based setup used to record target trajectories. Furthermore, an image of the different end-effector mounts together with the earmuffs and all there corresponding weights are depicted. (b) The top row illustrates the results during the tracking of a circular target trajectory (black) for different time instances (each time step 40 ms). The coloring of the trajectory generated by the WAM indicates which controller is dominant during the tracking of a particular part of the target. The bottom panel shows the corresponding responsibilities. The coloring here corresponds to a particular controller.



(a)



(b)

Figure 5.17: (a) This plot gives an overview of how the tracking performance improves over time, where each row represents a trial and each letter is written from left to right. Please note that the writing of the word "amarsi" is recorded one letter after the other but in the same region of the task space instead of one letter next to the other. The corresponding transition between two letters is not shown. (b) This graph, illustrates the difference in tracking distance between multiple weight configurations for several consecutive trials. The tracking distance shown, is the average distance during the tracking of the entire recorded trajectory. A new trial begins when the desired trajectory is repeated. When using no added weight (No haptic bal), the experiment measurements were performed for 5 trials, while for the other configurations measurements were taken during 14 trials.

6 Motion Planning and Control

In Section 5.2 a modular architecture was proposed, that uses multiple adaptive feedback controllers, presented in Chapter 4, to simplify the control task. However, these feedback controllers depend on a given desired trajectory, which needs to be tracked. In many robot applications, only a start position/velocity/acceleration and a goal position/velocity/acceleration are known, but not the path between both. Therefore, a planning strategy needs to be incorporated, which takes into account the restrictions imposed by the robot and its environment. In this chapter only, the application in underactuated control is considered, although it is possible to apply the combination of a planned trajectory and a feedback controller on a wide variety of tasks. In the first section, I will discuss the definition of an underactuated system and give an example that is used throughout this chapter. Next, the basic concepts behind trajectory planning are presented together with a planning algorithm. In Section 6.3, it is demonstrated how a planned trajectory is stabilized by introducing feedback with MACOP. Afterwards, I briefly describe the concept of a simulation-based control framework that combines the learning of the system dynamics, trajectory planning and trajectory stabilization. Finally, the similarities between this control framework and Reinforcement Learning are highlighted.

6.1 Underactuated control

An underactuated system is a system that has a lower number of actuators than degrees of freedom (DOF), of which the inverted pendulum is a classic example. This system has two DOF (cart position and pendulum angle) while only the position of the cart is actuated. In Chapter 4, I introduced such non-holonomic and underactuated control tasks (e.g., cart pole, double inverted pendulum) but considered only the balancing of the pendulum because the IMA controller is only able to track a

target. If the cart pole would be initialized in its stable downwards position, this controller will try to approach the upwards position by moving the cart as fast as possible in a single direction in order to reduce the distance to the target position. Due to the limited length of the rail and force that can be applied on the cart, it is impossible to reach this target. To approach the upward position, however, the pendulum needs to build up sufficient energy by swinging its pendulum back and forth, which means that the distance to the target sometimes increases, instead of only decreasing. A tracking controller, such as the one presented in Chapter 4, only tries to reduce the distance without taking into account the possible benefits of increasing the distance now and then.

Therefore, the main idea presented in this chapter is to use a planning algorithm to generate a trajectory in state space that can be tracked by a feedback control architecture such as MACOP. Such trajectory can guide the feedback controller towards the goal position. In robotics, such motion planning within dynamical constraints, where velocity, acceleration and force/torque bounds must be satisfied together with kinematic constraints, is called *kinodynamic planning*.

In the following section, I will briefly discuss the basic concepts behind trajectory planning and describe the planning algorithm that I have used.

6.2 Trajectory planning

When humans try to develop algorithms for controlling robots, there exists a rudimentary problem to translate a high-level specification of a task into a low-level description of how to move. As stated in LaValle (2006), the terms *trajectory planning* and *motion planning* are often used for such kinds of problems. Consider, for instance, the following classic scenario where one needs to move a couch through a house from the garage to the living room without colliding with walls or objects. The only high-level specification here is that the couch should be moved from the garage to the living room. The actual trajectory of the couch describing the six DOF (position and rotation) in a three-dimensional space, however, is not defined and should be found to solve the moving task. Planning algorithms are widely used in robotic applications. For instance, the autonomous cars competing in the DARPA Grand Challenge use planning algorithms to safely guide the car, driving at high speeds, over a rough terrain. Also, for industrial robotic purposes, such as the automated positioning of a dashboard into a car, similar planning algorithms are applied. In this section, the introduction about planning is limited to the basics needed to understand the algorithms used. For a more in-depth overview of planning algorithms, I would like to refer to the book Planning Algorithms by LaValle (2006).

In Chapter 4, a desired trajectory was defined by a sequence of desired plan-

t/system/robot states. However, more generally, a trajectory generated by a planning algorithm can be defined as a sequence of action-state pairs:

$$(\mathbf{y}(k+1), \mathbf{x}(k+1)) = f(\mathbf{y}(k), \mathbf{x}(k)), \quad \text{with } k = 1, \ldots, T \tag{6.1}$$

where $\mathbf{y}$ and $\mathbf{x}$ represent the state and actions, respectively, which conforms to the notation used in Chapter 4. The resulting action trajectory $\mathbf{x}(k)$ guides the plant from its initial state $\mathbf{y}(0)$ towards its goal state $\mathbf{y}(T)$, in which the completion time T (number of samples) to reach this goal state is not necessarily defined. The completion time can be a specification that limits the sets of possible solutions, but more often it is defined by the given solution itself.

In some cases, the state space in which a trajectory is planned is not static. For instance, when a robot needs to move through a dynamic environment where obstacles are moving, it becomes necessary to plan a new trajectory given the current state of the robot and its environment.

6.2.1 Configuration space

The *configuration space*, referred to as C-space, is a manifold in state space that contains all possible transformations (configuration) that could be applied to the robot in its environment. If we consider, for instance, a quad-copter robot that flies around in a three-dimensional environment the quad-copter's configuration is fully defined by a 3D position (X, Y, Z) and a 3D rotation (pitch, roll, yaw). A single point in C-space thus defines the 6D configuration of the robot. The use of a C-space provides a powerful abstraction level that transforms complicated transformations and robot models into a general problem of finding a path of a point on a manifold. As a result, most planning algorithms developed for such C-space can be applied on a wide variety of robotic applications. In order to allow such a planning algorithm to take into account restrictions in the robots movement, these restrictions need to be transformed to the configuration space as well. The subset of the C-space in which a configuration can move freely is called the *free space*. If we again consider the quad-copter example where obstacles are present, the *free space* of the robot does not contain configurations of the quad-copter in which a collision is possible. This C-space with obstacle representations can be visualized, in the actual environment, by growing the actual obstacle on each side with the maximum geometric size the quad-copter takes under any configuration. For a robotic manipulator with multiple degrees of freedom, however, this free space becomes much more complicated to visualize.

6.2.2 Planning algorithms

Over the years, many planning algorithms have been developed. Initially, simple algorithms were sufficient for the robots and tasks at that time. However, as the complexity of the robots and their tasks increased, new algorithms, that can handle many degrees of freedom, while still being computationally inexpensive, had to be developed. In this section, a brief overview is given of the most widely used types of planning algorithms.

- **Grid-based search:** Here a grid is placed over the C-space where each configuration is identified with a grid point. The robot can move from one grid point to one of its neighbors if the line between these points is fully included within the free space. Because of this discretization of the configuration space, the robot's actions are also discretized. Within such a context, a search algorithm (e.g., A-star) can be used to find an appropriate path from start to goal. The resolution of this grid is, of course, crucial in finding a feasible path. One of the main disadvantages of this grid-based approach is that the number of grid points in C-space increase exponentially with the number of C-space dimensions. So for high-dimensional planning problems, this approach is not appropriate.

- **Interval-based search:** Instead of placing a grid, the interval-based search generates a box paving covering the entire C-space. Each subpaving is represented by a node in a graph, and a path from start to goal can be found by a search algorithm. Another difference with the grid-based search is that the robot is allowed to move freely within a subpaving (no discrete actions). Similar to the grid-search, the number of boxes grows exponentially with the number of C-space dimensions.

- **Potential fields:** In this approach, values are placed over the configuration space in which the goal is appointed to a low value (attracting), and obstacles to an extremely high value (repelling). Finding a trajectory to the goal then becomes a gradient descent problem.

- **Sampling-based algorithms:** The basic idea of sampling-based algorithms is to draw a fixed number of samples from a uniform distribution over the C-space in which sample points that are not in free space are rejected. Two samples are connected with each other if their connection is completely within free space. Next, a shortest path search algorithm is applied to find a path from start to goal. These algorithms can be applied to tasks with a high-dimensional C-space because their running time does not depend exponentially on the dimension of the C-space. Furthermore, sampling-based methods are probabilistically complete because the probability of producing a solution approaches 1 as more time is spent during the trajectory search.

Due to the applicability of sampling-based algorithms to problems with hundreds of dimensions, these algorithms are currently considered as being state-of-the-art. Therefore, in the remainder of this chapter, a sampling-based algorithm, in particular the Rapidly-exploring Random Tree algorithm, will be used.

6.2.3 Rapidly-exploring Random Tree

Rapidly-exploring Random Tree (RRT), developed by LaValle and Kuffner Jr (2000), is an incremental sampling-based approach that quickly finds a feasible path between an initial state and goal state. Each node of the tree corresponds to a feasible robot configuration, and the edges between such nodes represent the necessary actions to go from one state to another. In practice, RRTs yield good task performance without any parameter tuning, which is the main reason for its wide use in robotic applications. Each iteration of the RRT algorithm has the following steps:

Step 1: The RRT is initialized with the start state $\mathbf{y}_0$ as first node of the tree.

Step 2: A random sample $\mathbf{y}_{\text{rand}}$ is drawn from a uniform distribution in C-space. If this sample does not belong to the free space, it is rejected and a new sample is drawn. To bias the search in the direction of the goal state $\mathbf{y}_K$, the goal state is chosen with a certain probability instead of a random sample.

Step 3: Given $\mathbf{y}_{\text{rand}}$ the nearest node in the tree $\mathbf{y}_{\text{near}}$ is searched. A Euclidean distance metric is used, although more appropriate metrics are possible as well. This is shown in panel A of Figure 6.1.

Step 4: The tree is extended from $\mathbf{y}_{\text{near}}$ towards $\mathbf{y}_{\text{rand}}$ by applying a certain control command $\mathbf{x}$ for a single time step Δt. This control command can be calculated by using model information of the robot or by selecting several random actions and selecting the one that approaches $\mathbf{y}_{\text{rand}}$ as close as possible. However, choosing a random action might not be optimal. Therefore, often an action is chosen from a fixed set of policies (i.e., a fixed set of commands) (Frazzoli et al., 2002). As a result of the chosen action $\mathbf{x}$, a certain state $\mathbf{y}_{\text{new}}$ is reached which is an approximation of $\mathbf{y}_{\text{rand}}$. Panel B of Figure 6.1 illustrates these steps.

Step 5: Add sample $\mathbf{y}_{\text{new}}$ together with the corresponding action $\mathbf{x}$ to the tree.

Step 6: Go back to Step 2 (shown in panel C of Figure 6.1) until $\mathbf{y}_{\text{new}}$ is close enough to the goal state $\mathbf{y}_K$.

When the RRT reaches the goal state $\mathbf{y}_K$ within a predefined distance, the shortest path of actions within this tree can be used to approach this goal state from the start state $\mathbf{y}_0$.

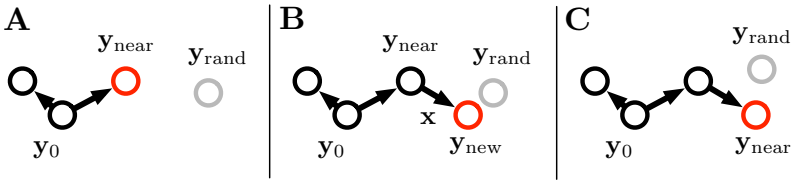


Figure 6.1: Illustration of how the nearest node $\mathbf{y}_{\text{near}}$, to a uniformly drawn sample $\mathbf{y}_{\text{rand}}$, is extended towards a new node $\mathbf{y}_{\text{new}}$ by applying an action $\mathbf{x}$ in a Rapidly-exploring Random Tree. $\mathbf{y}_0$ is the start state or root of the RRT.

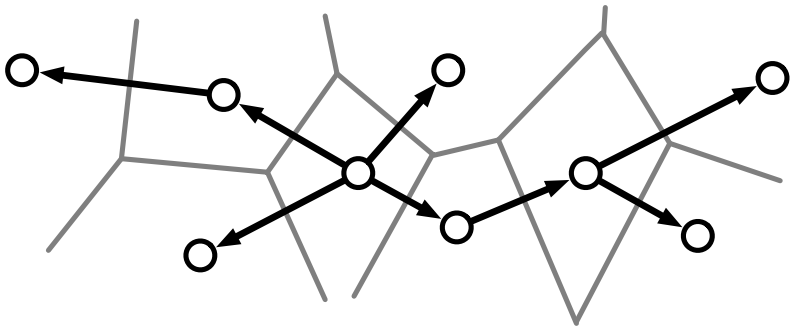


Figure 6.2: Illustration of a RRT with the associated Voronoi diagram. The search for new sample points is biased towards larger regions, that makes the RRT algorithm efficient in covering the C-space.

6.2.3.1 Voronoi bias

A Voronoi diagram in mathematics is an approach to divide a space (in this chapter, the C-space) into a number of regions, where the regions are chosen based on the distance to an RRT node. All samples chosen within a region belong to a single nearest RRT node. The probability that a tree node is selected to be extended depends, therefore, on the volume of its Voronoi region. As a result, the search for new samples is biased, hence the name Voronoi bias, towards larger Voronoi regions, that allows the RRT to cover the C-space efficiently (LaValle, 2006).

6.2.3.2 Task space control

The IMA controller introduced in Chapter 4 requires a desired trajectory that needs to be tracked. As I have mentioned at the beginning of this chapter, such a trajectory is not always known. For instance, a mobile robot with differential wheels needs to find a path from one office to another without colliding with walls or doors. For

such a basic robot (low-dimensional), it is possible to map the C-space back onto the task space uniquely by increasing the size of each obstacle with the maximum distance any robot configuration can have. As a result, in such a constrained task space, a trajectory can be planned and applied onto the robot by an inverse dynamic/kinematic controller without the need of having to check the configuration of the robot.

For high-dimensional robots with many DOF, such an approach becomes infeasible because for any position in task space there exist an infinite number of possible configurations. Mapping the C-space back onto the task space to plan a trajectory in task space is therefore not useful. Consider, for instance, the inverse kinematic learning task in Sections 4.3.1 and 5.2. Here, a desired trajectory was defined that the controller should learn to track, neglecting any constraints on the actual pose of the robot during its motion. Such a task becomes much harder to solve if a task is considered where the robot manipulator should reach for an object that is located on the other side of a wall that separates the robot from the target object. Only by reaching through a small hole in the wall will the robot be able to approach the target. In order to solve this problem, a trajectory should be planned that guides the end-effector towards the target and avoid any robot configuration that leads to collisions between the manipulator and the wall. Shkolnik and Tedrake (2009) solved this problem by creating an RRT in task space (TS-RRT) where samples are drawn in task space instead of C-space. After drawing a sample, the nearest neighbor in task space is searched for, from which the tree can be extended by applying corresponding joint angles that approximate the task space sample by means of an inverse kinematic model. This inverse kinematic model uses the joint configuration of the nearest tree node to calculate the closest configuration. If the corresponding joint configuration is collision free, the new node, together with the corresponding configuration (and torque if joints are torque controlled), is added to the tree. This method proved to be very efficient and therefore scalable to a large number of dimensions.

6.2.4 Reachability Guided RRT

Although it is possible to use the RRT planning algorithm to plan a trajectory in task space, the main goal of this chapter is to achieve underactuated control. My experience with RRT algorithms showed, however, that the number of iterations needed by a naive RRT implementation is extremely high, especially when trying to find a path for a cart pole system with a limited rail length which is naturally extremely unstable. When using inappropriate distance metrics (e.g. Euclidean distance), new samples tend to be chosen more at the boundaries defined by the dynamical constraints of the system instead of achieving any progress in expanding the tree towards unvisited regions of the state space. Therefore, I implemented

a more sophisticated strategy that heuristically changes the Voronoi bias to more feasible sampling points.

In Shkolnik et al. (2009) an extension, called Reachability Guided - RRT (RG-RRT), of the naive RRT algorithm was proposed. This extension is based on the observation that a Euclidean distance metric is inappropriate for problems with kinodynamic constraints. For instance, when the amount of force that can be applied onto the pendulum cart is limited, some points in state space might be very hard to reach, although they lie very close to each other in state space. Therefore, many researches have searched for other distance metrics that are more suitable for tasks with kinodynamic constraints yielding several flavors of RRT-based planning algorithms, each with their advantages and disadvantages. For instance, in Glassman and Tedrake (2010), an affine LQR is placed around each sample point and the cost-to-go is used as distance metric. Another modification of the RRT algorithm reduces the size of the Voronoi regions in the neighborhood of obstacles so that the Voronoi bias is shifted towards samples with no neighboring obstacles (Yershova and LaValle, 2007).

For most control tasks, the actions applied on a dynamical system are limited to a predefined range. If, again, the inverted pendulum task is considered, the force that can be applied on the cart is limited to the range $[-F_{\max}, F_{\max}]$. When drawing a new state space sample it is, for instance, not useful to choose a nearest node from which it is impossible to reach the nearest node, given the minimum and maximum force that can be applied on the cart. Given this insight, the naive RRT algorithm described in Section 6.2.3 is modified as follows:

Step 1: The first node of the tree contains the start state $\mathbf{y}_0$ together with the states that can be reached by applying $\pm F_{\max}$. In order to calculate these states, the initial state is applied to a forward model for each force limit, yielding a set of states, called the *reachable set*. Each node thus contains a parent state and a reachable set $\mathbf{R}$ of states.

Step 2: A random sample $\mathbf{y}_{\text{rand}}$ is drawn from a uniform distribution in state space. If this sample does not belong to the free space, it is rejected and a new sample is drawn. To bias the search in the direction of the goal state $\mathbf{y}_K$, the goal state is chosen with a certain probability instead of a random sample.

Step 3: Next, the Euclidean distance is calculated between $\mathbf{y}_{\text{rand}}$ and every parent state and reachable state within every node. The distance to the closest parent state $\mathbf{y}_{\text{near}}$ is compared to the distance to the closest reachable state $\mathbf{R}_{\text{near}}$. If the distance to $\mathbf{y}_{\text{near}}$ is smaller than the distance to $\mathbf{R}_{\text{near}}$, the sample is rejected and a new sample is drawn (Step 2). This rejection is shown in Panel A of Figure 6.3. Otherwise, the parent state of the node corresponding to $\mathbf{R}_{\text{near}}$ is called $\mathbf{y}_{\text{near}}$ (Panel B of Figure 6.3).

Step 4: As illustrated in Panel C of Figure 6.3, the tree is extended from $\mathbf{y}_{\text{near}}$

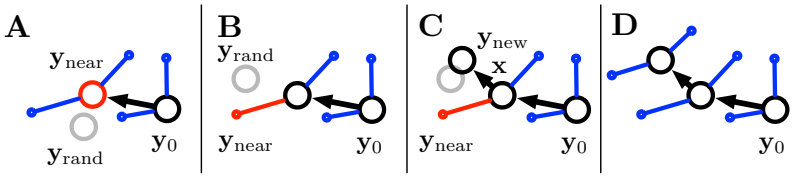


Figure 6.3: Illustration of how an RG-RRT is extended towards a uniformly drawn sample $\mathbf{y}_{\text{rand}}$ (gray big circle) in state space. A parent state is indicated by a big black circle, while the states in the reachability set are indicated by small blue points. The action that yields such reachable states belongs to the force limits $\pm F_{\text{max}}$. The nearest state to $\mathbf{y}_{\text{rand}}$ is indicated by a red color.

towards $\mathbf{y}_{\text{rand}}$ by applying a certain control command $\mathbf{x}$ for a single time step Δt. A control command is chosen from a set of predefined control commands (within force range), instead of calculating the necessary force with an inverse model or choosing a random action. As a result of the chosen action $\mathbf{x}$, a certain state $\mathbf{y}_{\text{new}}$ is reached that is an approximation of $\mathbf{y}_{\text{rand}}$. Furthermore, the reachable set $\mathbf{R}$ is calculated for this new state $\mathbf{y}_{\text{new}}$.

Step 5: Add sample $\mathbf{y}_{\text{new}}$ together with the corresponding action $\mathbf{x}$ and $\mathbf{R}$ as a node to the tree. Demonstrated in Panel D of Figure 6.3.

Step 6: Go back to Step 2 until $\mathbf{y}_{\text{new}}$ is close enough to the goal state $\mathbf{y}_K$.

As demonstrated by Shkolnik et al. (2009), there is a clear performance increase when using an RG-RRT compared to a naive RRT implementation, because the sample rejection process causes the tree to grow in the direction of reachable and unexplored regions. The Voronoi bias is thus adjusted in order that nodes that are better suited to explore the state space are chosen more often to extend the tree from.

6.2.4.1 Bi-directional extension

When the RG-RRT algorithm is applied on a forward model of the cart pole task, an action trajectory will be found that achieves a swing-up of the pendulum. However, because of the strong constraints on the task (limited rail length, limited force) it still takes a while to achieve a trajectory that approaches the target[1]. In order to accelerate the search, the RG-RRT implementation is extended to a bi-directional

[1]The target here is the upwards pendulum position and cart centered on the rail

version, where an additional tree is grown from the goal state towards the start state. For this extension, I used the following procedure based on Kuffner Jr and LaValle (2000):

Step 1: Initialize the two trees, $\mathcal{T}_\mathrm{A}$ and $\mathcal{T}_\mathrm{B}$ with the start and goal state, respectively.

Step 2: Expand one tree until it has 2 more nodes than the other, after which the other tree is expanded. As a result, $\mathcal{T}_\mathrm{A}$ and $\mathcal{T}_\mathrm{B}$ are grown in turn. The samples of $\mathcal{T}_\mathrm{A}$ are biased (with a certain probability the target state is sampled instead of a random sample) towards the goal state and those of $\mathcal{T}_\mathrm{B}$ in the direction of the start state.

Step 3: Both trees will grow towards each other until one tree reaches its target state or the other tree within a predefined distance. If $\mathcal{T}_\mathrm{A}$ reaches the goal state, the corresponding nodes describe a path of states and forces. If $\mathcal{T}_\mathrm{B}$ reaches the start state, the resulting trajectory needs to be reversed before it can be applied on the cart pole setup. However, when both trees reach each other, the corresponding trajectories need to be connected with each other (after reversing the path of $\mathcal{T}_\mathrm{B}$)

Before applying this algorithm onto the cart pole swing-up task, the following Euclidean distance metric is defined:

$$D = \|\mathbf{w}_\mathbf{y}(\mathbf{y}_K - \mathbf{y})\|, \tag{6.2}$$

with $\mathbf{w}_\mathbf{y}$ a weight vector defining the importance in the distance measurement. For the cart pole task I used the following weights: $[w_\theta, w_{\dot{\theta}}, w_x, w_{\dot{x}}] = [0.6, 0.025, 0.35, 0.025]$. The target state $\mathbf{y}_K$ depends on the nature of the tree (forward or backward).

Please note that for now the actual differential equation is used as a forward model. However, in Section 6.4 more details are given about how such a forward model can be learned. Moreover, I will discuss how a forward model can be used to approximate the backward dynamics needed to grow the backward tree.

In Figure 6.4, a bi-directional RG-RRT (bi-RG-RRT) planner is applied on the cart pole task where a forward and backward tree are connected with each other, resulting in a trajectory that describes the swing-up motion of the pendulum. Based on the shown state space samples drawn (gray dots), one can observe the chosen limits. For instance, the limits of the cart position correspond to the length of the rail.

Both trees together contain 816 nodes of which 95 describe the resulting trajectory. These numbers of course depend on the complexity of the task and the defined limits in which samples should be drawn. Furthermore, the sample rate defines the time between 2 nodes. Therefore, a slower sample rate could yield less nodes than are necessary to reach the upwards position. In this case, however, the

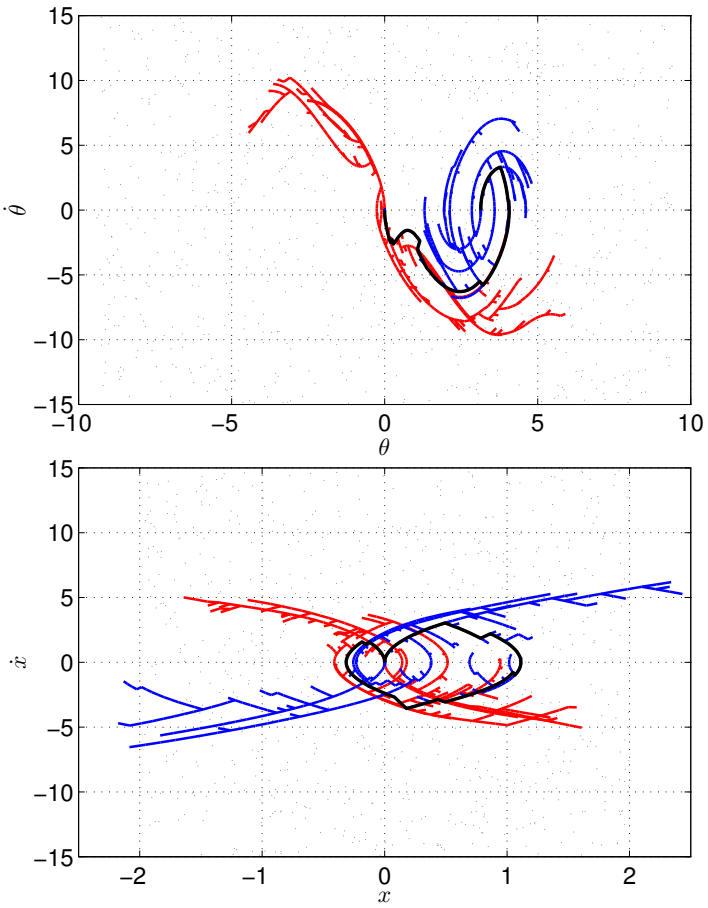


Figure 6.4: Both plots illustrate an example result of running bi-directional RG-RRT on the cart pole task. The forward tree (blue) starts at $\mathbf{y} = [\theta, \dot{\theta}, x, \dot{x}] = [\pi, 0, 0, 0]$, while the backward tree (red) starts at the goal state $[0, 0, 0, 0]$. The resulting trajectory is shown by the black curve. All the drawn samples during the creation of this bi-directional tree are represented by gray dots. As shown, they are distributed within predefined limits. The actual differential equation is used as a forward model.

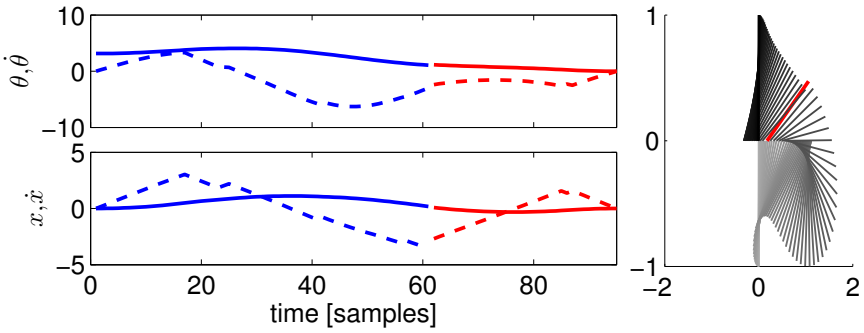


Figure 6.5: Both left plots show the resulting trajectory where the color indicates the contribution of a particular tree (red: backward tree, blue: forward tree). The solid and dashed lines represent θ, x and $\dot{\theta}, \dot{x}$ respectively. The right plot visualizes the resulting state trajectory allowing for the motion of the pendulum to be better interpreted. The entire motion, from the initial downwards position to the upwards position, is indicated by a gray to black coloring of the pendulum. The point at which the trees are connected is indicated by a red colored pendulum.

resulting trajectory will be less smooth and therefore not necessarily applicable on a real experimental setup.

6.3 Trajectory stabilization

Applying an RRT generated trajectory onto the used forward model of the pendulum will cause its states to evolve towards the target state. Despite that, when a bi-directional RRT is used, there exists a sudden, often small, jump in the state and force of trajectory where the two trees connect with each other. Such a gap will cause the swing-up of the pendulum to fail, even on an ideal pendulum where no friction exists. For instance, in Figure 6.5, the resulting state trajectory corresponding to Figure 6.4 is shown. The moment at which the trajectory jumps from the forward to the backward tree is indicated by the color of the trajectory. When drawing the corresponding pendulum motion (right plot of Figure 6.5) this gap is visible after the red pendulum, where there exists a sudden jump in the smooth trajectory of the tip of the pendulum. Depending on the distance of this gap, the associated force trajectory will cause the pendulum not to reach the upwards position. When applying a trajectory generated by an RRT, be it bi-directional or not, on a real pendulum

setup, the pendulum will also not reach the desired target. Every trajectory generated by an RRT is an open loop trajectory, which means that the control is unable to compensate for small modeling inaccuracies or control noise. In order to solve this problem, the trajectory needs to be stabilized allowing that feedback information can be taken into account. There are several approaches to solve this stabilization. For instance, in Tedrake (2009), a sampling-based approach, called LQR-Tree algorithm, is proposed, that stabilizes the trajectory with local LQR feedback. Each LQR grabs initial conditions within its neighborhood and pulls them towards the local target from which another LQR begins pulling. Furthermore, the size of the basin of attraction of every added LQR is calculated so that new samples can be chosen outside this attractor region yielding a sparse tree of local controllers. A similar approach (Maeda et al., 2010) schedules LQR controllers incrementally in an attempt to connect the states generated by a bi-directional RRT, and it is verified that every connected state is within a controllable region of an LQR controller. Although these techniques have demonstrated that they perform really well, they depend on model information (i.e., mass, friction and length of pendulum and cart) for their LQR design.

6.3.0.2 Introducing feedback with MACOP

In order to stabilize the trajectory without model information, a different feedback control approach is needed. Interestingly, the idea of using a set of LQR controllers to linearly approximate the feedback control of the non-linear control task is similar to the underlying concept of MACOP. However, as demonstrated in Section 5.2, MACOP can rely on a set of non-linear controllers, which allow a lower number of controllers to perform as well as a larger number of linear controllers.

As shown in Figure 6.6, MACOP can be applied by learning the necessary stabilization corrections on the force trajectory $F(t)$ generated by the bi-RG-RRT (panel A). The complexity of the needed models is quite low because Tedrake (2009) and Maeda et al. (2010) have shown, among others, that a set of linear controllers is sufficient to stabilize the trajectory given $F(t)$. For an online learning approach, such as MACOP, the used force limiter, that bounds the resulting force to $F_{\max}$, introduces a difficulty in learning the trajectory stabilization. MACOP uses correlations between its own introduced force values and the corresponding responses in state space. Unfortunately, the generated force values are added to $F(t)$ and then limited, which removes correlations when $F(t)$ reaches its limits. One approach to solve this is to give MACOP information on its actual contribution to the final force value F^* after the limiter. For example, if $F(t) = F_{\max}$ MACOP's contribution after the limiter should be 0 N. Another approach, that will be used in the remainder of this chapter, is shown in panel B of Figure 6.6. Instead of learning the necessary contributions to the planned force trajectory $F(t)$, MACOP is used to learn the entire control signal based on the generated state space trajectory. Because

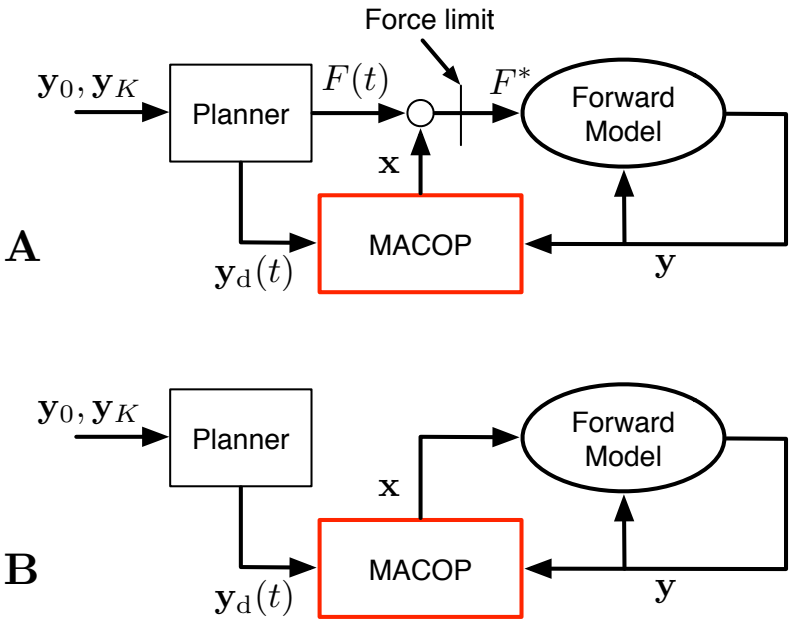


Figure 6.6: Panel A depicts a control strategy where MACOP learns the necessary contributions $\mathbf{x}$ to the control force $F*$ in order to stabilize the state space trajectory. This trajectory is generated by the bi-RG-RRT algorithm based on an initial state $\mathbf{y}_0$ and target state $\mathbf{y}_K$. Panel B represents a control strategy where MACOP learns the control signals only based on the generated state space trajectory. Instead of using a forward model it is also possible to use the actual cart pole setup. The red color specifies that a system is adapted during the process.

Table 6.1: Network parameters for the cart pole task

Parameter	Value	Parameter	Value
N	200	α	0.01
ρ	1	f_i^r	0.7
δ	1	f_b^r	0.2
λ	$1-10^{-6}$	γ	1

each controller within MACOP contains an output limiter, the output of MACOP $\mathbf{x}$ is already bounded to the imposed force limits.

In the following experiments, I used a MACOP configuration that contains 6 IMA controllers. The same network parameters, shown in Table 6.1, were used for every IMA controller. For this task, I also wanted to visualize the controllers responsibility, similar to the actuator example in Figure 5.7. Therefore, MACOP is configured in such a way that the tiling occurs in the space span by $\sin(\theta)$ and $\cos(\theta)$. So in Equation (5.6), $\mathbf{y}(t) = [\sin(\theta), \cos(\theta)]$ and $\widehat{\mathbf{x}}(t)$ is ignored, allowing for this two dimensional space to be tiled in wedge shaped parts. As a result, the use of a particular controller only depends on the current angle of the inverted pendulum.

In Figure 6.7, the resulting effect of trajectory stabilization is demonstrated and compared with the pendulum's response to the open loop force trajectory obtained during planning with bi-RG-RRT. The red colored pendulum, which is controlled by the open loop force trajectory, follows its own state space until it reaches the gap that was caused by connecting the forward and backward tree. After this point, the actual pendulum trajectory starts to diverge from the planned trajectory. However, by learning the closed loop force control only based on the state space trajectory, for example with MACOP, it becomes possible to stabilize trajectory in such a way that the goal position is reached. The blue pendulum in Figure 6.7 demonstrates the resulting motion when such closed loop control is delivered by MACOP in accordance to panel B of Figure 6.6. The corresponding state space trajectory shows small differences with the desired one (provided by planner). However, this desired state space trajectory is not a realistic trajectory because of the sudden jump caused by connecting both trees. As I have mentioned in Chapter 4, the tracking of the desired trajectory is the result of an accurate inverse model, because the training error is taken at the output, and not the input, of each controller. So, each controller within MACOP learns a local inverse model of the cart pole dynamics, that tries to track the desired trajectory as good as possible with a resulting physically valid trajectory. The bottom graph of Figure 6.7, demonstrates the difference between the planned force trajectory (blue) and the one learned by MACOP (red). Clearly,

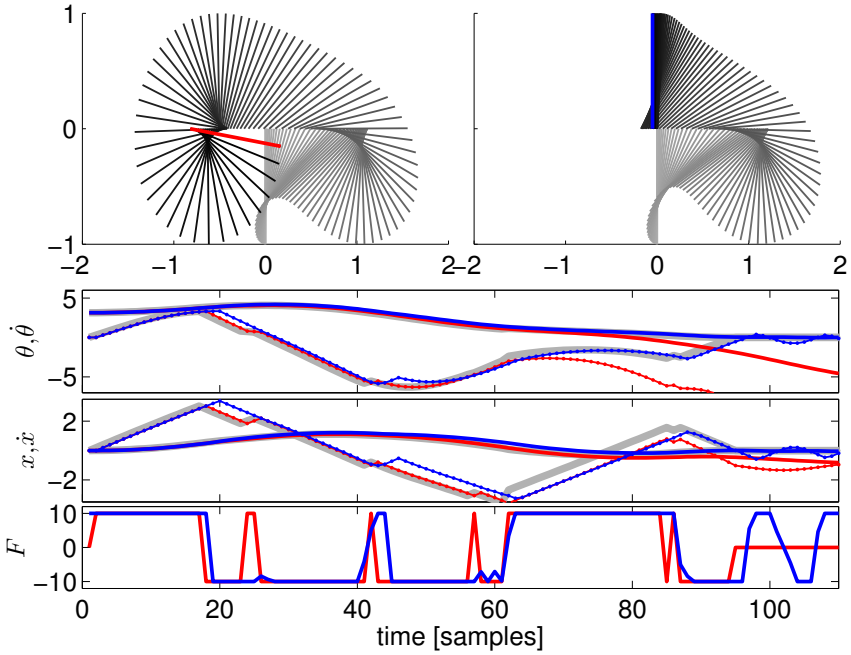


Figure 6.7: This plot demonstrates the difference between open loop control (red pendulum) and closed loop control (blue pendulum). The state space trajectory and open loop control signals are produced by the bi-RG-RRT algorithm. The closed loop control is learned based on the planned state space trajectory. The top two plots represent the resulting pendulum motion for each control strategy. The graphs underneath illustrate the actual evolution of the states compared to the planned trajectory (thick gray line). The dotted trajectories represent the angular velocity, or the velocity of the cart. The coloring of the curves corresponds to the control strategy used. The bottom graph illustrates the difference in control signals.

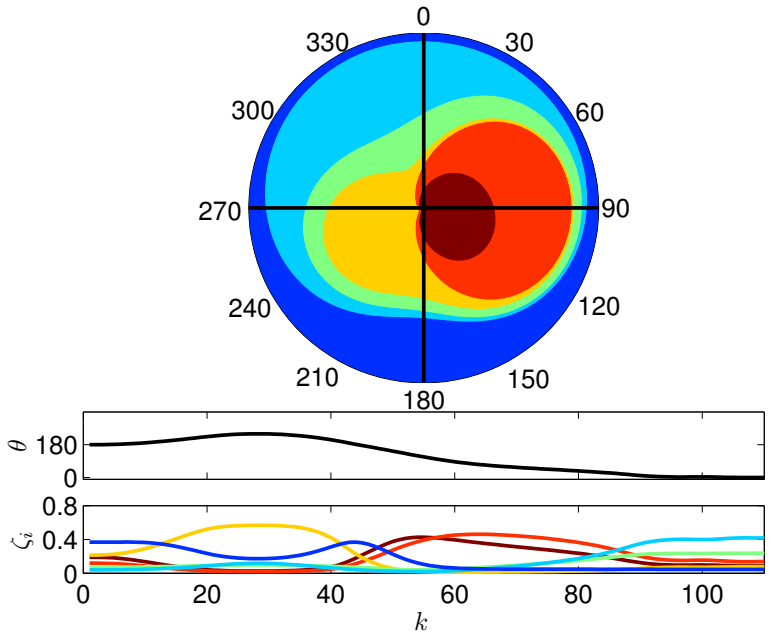


Figure 6.8: This plot illustrates how the responsibilities ζ_i are distributed for a given angle of the pendulum. The middle plot shows the angle θ during the swing-up phase while the bottom plot shows the corresponding scaling factors.

MACOP generates a more continuous set of control signals, although most of it resembles a bang-bang control behavior.

In Figure 6.8 the fractions of all scaling factors are shown for all possible pendulum angles. This gives an indication of the responsibility of a particular controller for a particular angle. The two bottom plots, illustrate these scaling factors during the course of the swing-up trajectory. MACOP is following the planned state space trajectory, but the trajectory is not visiting all angles equally. Some angles are never reached, others are visited multiple times and some are visited slower than others. Due to MACOP's mixing heuristic, the responsibilities will be shorter in time for the more frequently visited angles, compared to the others. This is the reason why the distribution of the scaling factors, show in top plot of Figure 6.8, is shifted to the right.

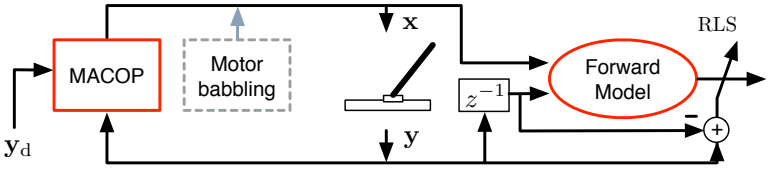


Figure 6.9: Illustration of how a forward and inverse model (MACOP) can be trained online and at the same time. The task space position is indicated by $\mathbf{y}$, while $\mathbf{x}$ denotes the angular joint positions. $\mathbf{y}_d$ is an arbitrary target trajectory that needs to be tracked. Ideally, this trajectory explores the full range in which the robot can move. The red color specifies that a system is adapted during the process.

6.4 Forward model learning

When using any planner algorithm, each addition to the path should be evaluated on a model of the robot. Until now, in our cart pole example application, the actual differential equation was used as a forward model. The main assumption in this dissertation is, however, that no prior model knowledge is available. Therefore, a model needs to be learned based on data recorded by interacting with the robot.

As shown in Figure 6.9, the input of the model is the previous state space position together with the applied action. The output, on the other hand, is the change in state value. By using, for instance, motor babbling (i.e., random action) on the cart pole setup, the necessary training data is generated to learn a model. Also, MACOP can be applied with an arbitrary target trajectory to generate enough training data to learn a forward model.

The accuracy of the model depends on how the training data is distributed over the state space. If the entire state space is reachable by applying motor babbling, the model will perform well in all regions of this state space. However, if the magnitude of the random actions is only sufficient to reach a part of the state space, the model will be inaccurate in the other regions.

The model accuracy also depends on the amount of training data available. Some modeling approaches, such as Gaussian Process Regression, are able to create a model with a very limited amount of training data. Other approaches, such as Locally Weighted Regression, Locally Weighted Projection Regression and ESNs, need much more data to get a useful model. In Ilin et al. (2004) a non-linear dynamical factor analysis (NDFA) algorithm is proposed that can create a very good non-linear dynamical state space model of the system with enough statistics of the system dynamics. As an example of learning a forward model, I will discuss an

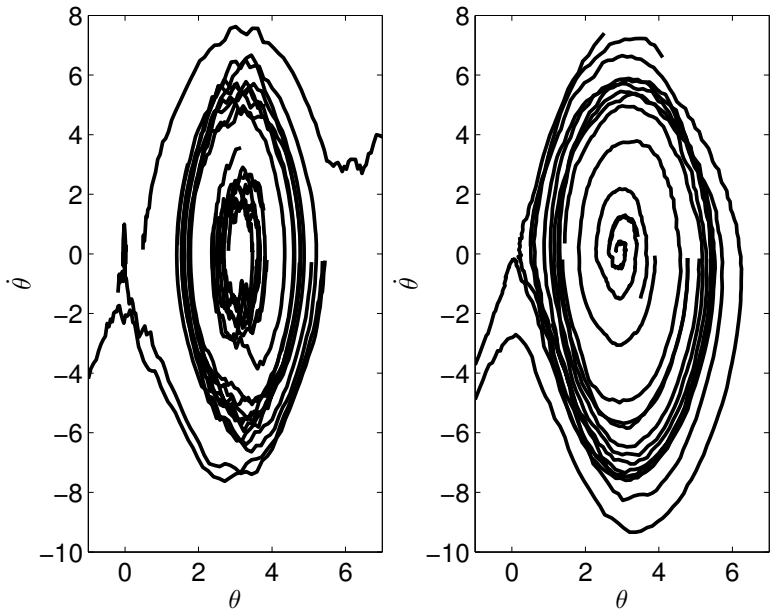


Figure 6.10: The left and right plot illustrate a phase plot of the pendulum dynamics. The left plot represents the reference (known) model, while the right plot represents the learned ESN model. By initializing the models for different states and recording their evolution, the well known shape of the pendulum dynamics becomes visible.

ESN-based approach that uses the RLS algorithm presented in Section 2.1.5.4.

When training an ESN with the network parameters shown in Table 6.2, it becomes possible to learn an ESN model based on observations made on the real plant or reference model. Due to the ESN recurrent connections there exists a transient behavior when the model input is switched. When learning a dynamic mapping, such transient behavior is not desired. Therefore, the network is trained in such a way that this transient behavior is reduced. All recorded input-output examples are randomly shuffled, forcing the ESN to handle abrupt changes in its input. Furthermore, before a new training example is provided to the ESN, the network states are reset to their initial value. After training the ESN, the network can be excited with any initial state and force, within a meaningful range. As illustrated in Figure 6.10, the ESN indeed has learned to produce a similar dynamic relationship between its input and output, when compared to the reference model. As shown in Figure 6.11, the states evolve initially the same as the reference model.

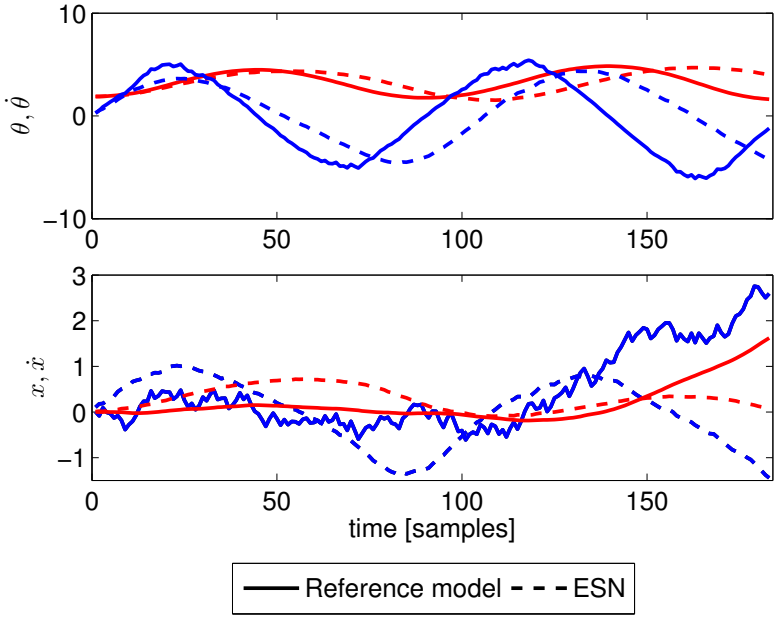


Figure 6.11: This Figure compares the state evolution of the reference model (solid line) and the ESN model (dashed line), for a given initial state. The top plot illustrates the evolution of θ and $\dot{\theta}$, denoted by the red and blue lines, respectively. In contrast, the bottom graph shows x and $\dot{x}$, denoted by the red and blue lines, respectively.

Table 6.2: Network parameters of the forward modeling task

Parameter	Value	Parameter	Value
N	400 neurons	f_i^r	0.9
ρ	1	f_b^r	0.1
γ	1		

However, the newly calculated states are fed-back to the input of the model. So, after a while, small errors will have their effect on the future evolution of the states. I experienced that modeling the state transitions of the cart position and velocity is rather difficult with an ESN. Arguably, the non smooth nature of the corresponding manifold is hard to be modeled with an ESN. Therefore, it is advisable to use a modeling approach that learns to fit a manifold that is not intrinsically smooth.

When a bi-directional RRT is used, a backward tree is grown starting from the target state. Growing such a tree backwards requires the forward model to be reversible. Furthermore, as mentioned before in this section, the model should also be trained on states in the neighborhood of the target. Otherwise, the backward tree will not contribute much to the planning.

6.4.1 Reversible forward model

When using a bi-directional RRT, the system dynamics need to be simulated in reverse, in order for the tree that starts from the goal state to be able to determine a new state $\mathbf{y}_{\text{new}}$ based on an action $\mathbf{x}$. Given a deterministic differential equation describing the dynamical system, it is possible to invert time if there exists a one-to-one mapping between the forward and reversed time dynamics (Lamb and Roberts, 1998). The forward dynamics can be written as follows:

$$\mathbf{y}(t+\Delta t) = \mathbf{y}(t) + \frac{d\mathbf{y}(t)}{dt}\Delta t \tag{6.3}$$

Given the fact that the used integration time step Δt is chosen as relatively small, we can approximate (no energy loss) the dynamics backwards in time as follows:

$$\mathbf{y}(t) = \mathbf{y}(t+\Delta t) + \frac{d\mathbf{y}(t+\Delta t)}{dt}\Delta t \tag{6.4}$$

$$\approx \mathbf{y}(t+\Delta t) + \left(-\frac{d\mathbf{y}(t)}{dt}\Delta t\right). \tag{6.5}$$

This means that the trained forward model from Section 6.4 can be used for the backwards RRT, $\mathcal{T}_{\mathrm{B}}$, to approximate a new state given a certain control action.

6.5 Simulation-based control framework

As I have discussed in Section 6.4, when learning a forward model, it is possible that the motor babbling will never cause the pendulum to reach the upper regions of its task space, which means that the forward model will never be accurate in such unexplored regions. However, planning and stabilizing a trajectory based on an inaccurate forward model will yield an inaccurate controller that tries to move towards such regions. These control efforts yield new data that can be used to improve the accuracy of the forward model. By iteratively updating the forward model, on which the planner can generate a trajectory that the controller can learn to follow, the controller will eventually be able to solve the task. This strategy is not new and has been validated by other approaches before (Deisenroth and Rasmussen, 2011). Based on this concept, a control framework is proposed in Figure 6.12 that uses an RRT as planner and MACOP as controller. The previous sections within this chapter demonstrated how each part within the shown framework works separately. Unfortunately, the validation and evaluation of the fully interconnected framework remains to be done and will be a focus in future work.

6.6 Link with Reinforcement Learning

Most algorithms within this dissertation are supervised learning algorithms and need training examples that consists of example input-output pairs in order to learn the underlying relationship between the input and output, or vice versa. For instance, in the above used cart pole task, the input denotes the state space observations, while the output is represented by the actions/forces applied onto the cart. For most robotic tasks, it is rather difficult to obtain such example data, especially the corresponding output to a certain input. As shown in Chapter 4, one can interact with the cart pole system to collect the necessary data in such a way that an incremental supervised learning algorithm becomes applicable. However, inspired by behavioral psychology, another approach, called *Reinforcement Learning*, was developed that does not need training examples of the corresponding actions. Instead, this unsupervised learning strategy uses a different kind of feedback signal that encodes how well the applied actions are contributing to the desired objective. As shown in Figure 6.13, the RL algorithm learns the appropriate actions based on a reward signal, given the current state of the pendulum, so that the cumulative reward is maximized. Maximizing a reward function at a future moment in time

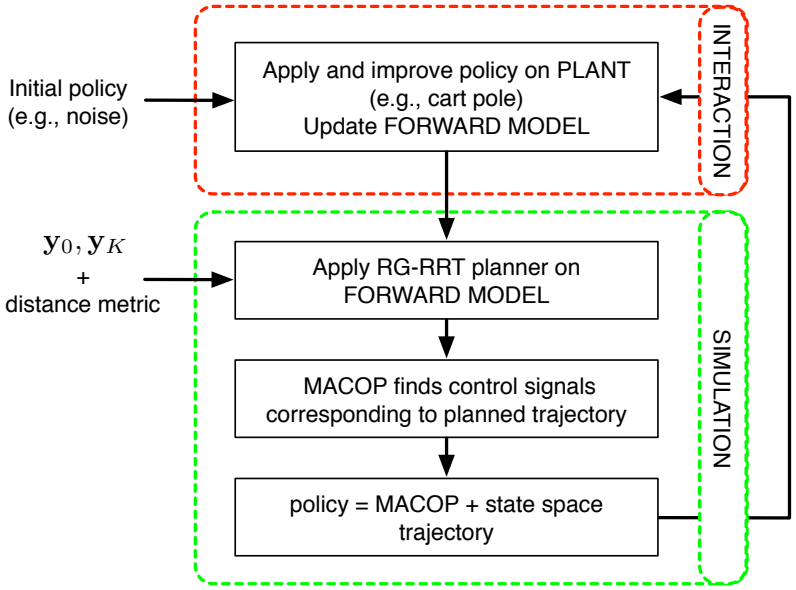


Figure 6.12: Illustration of the proposed simulation-based control framework that learns a control policy iteratively, without any prior model information. As shown, the framework only needs the initial state $\mathbf{y}_0$, the goal state $\mathbf{y}_K$, an appropriate distance metric (e.g., weighted Euclidean distance) and an initial exploration policy (e.g., noise). The *interaction* phase consists of applying the policy on the real system and updating a forward model. During the *simulation* phase, the policy is optimized based on this forward model.

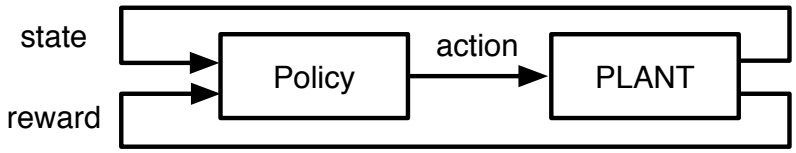


Figure 6.13: Illustration of a basic reinforcement learning framework.

sometimes requires sub-optimal actions to be performed in the present. This also accounts for the underactuated cart pole problem where a swing-up phase is necessary before being able to reach the upwards position. RL algorithms have shown (Sutton and Barto, 1998) to be well suited for such problems where a trade-off has to be made between short-term and long-term reward. Thus, there also exists a planning problem in which a plan needs to be determined that optimizes the rewards. As mentioned in LaValle (2006), the term *reinforcement planning* may thus be as appropriate as *reinforcement learning*. The general framework of reinforcement learning is based on concepts of *dynamic programming* and is therefore often referred to as *neuro-dynamic programming*[2] (Bertsekas and Tsitsiklis, 1995). The term dynamic programming was invented by Bellman (1952) and relates to the simplification of a decision by breaking it up into a plan or sequence of sub-decisions over time. For each time instance, i a *value function* V_i, is calculated depending on the state, $\mathbf{y}$, at that time. Given the following recursive Bellman equation it becomes possible to calculate the optimal V_{i-j}, with $j > 0$ backwards from the present V_k by maximizing the value gain recursively from $i-1$ to i with $i = k, \ldots, 1$, if a decision is made:

$$V(\mathbf{y}_{i-1}) = \max_{\mathbf{x}_{i-1}} \{R(\mathbf{y}_{i-1}, \mathbf{x}_{i-1}) + \beta V(\mathbf{y}_i)\}, \tag{6.6}$$

where $R(\mathbf{y}_{i-1}, \mathbf{x}_{i-1})$ denotes the reward/payoff of taking action $\mathbf{x}_{i-1}$ in state $\mathbf{y}_{i-1}$). A discount factor that determines the influence of the corresponding decision at time i is indicated by $\beta \in]0,1[$. Furthermore, this equation is only valid under the constraints that action $\mathbf{x}_i$ belongs to a set of possible actions that depend on the corresponding state $\mathbf{y}_i$, and the state $\mathbf{y}_i$ itself depends on the previous state $\mathbf{y}_{i-1}$ and action $\mathbf{x}_{i-1}$. When the Bellman equation is applied, and V_1 has been reached, the optimal sequence of decisions can be retrieved from the value maximization.

The Bellman Equation (6.6) defines a family of value-function-based approaches for which there exist many reinforcement learning algorithms. A sub-set of these value-based approaches are called *simulation-based* approaches. Often, the transition from one state to the next, given an action, is modeled by retrieving statistics from the actual plant (by interacting with it). Given these statistics, a distribution

[2]The term neuro points to a class of function approximators that can be used to represent plans or value functions among others.

can be calculated from which samples can be drawn. The term *simulation-based* thus comes from the fact that, for instance, a Monte Carlo simulator is used to draw the necessary state transitions during planning. According to LaValle (2006), there exists, in general, three phases:

- **Learning phase:** By interacting with the plant, a forward model is created that models the state transition given the current state and a new action.
- **Planning phase:** A value iteration approach is applied on the learned model (i.e., simulated) to compute a feedback plan that approaches the optimal reward. For episodic problems, a policy iteration approach is often used where a policy is evaluated after an episode and updated according to the newly determined sum of rewards collected during an episode.
- **Execution phase:** The computed feedback plan, called policy, is applied on the real plant.

Given the simulation-based control framework proposed in Section 6.5, and shown in Figure 6.12, one can identify the same phases. In the *learning phase* a forward model is learned. However, instead of retrieving statistics and calculating the corresponding distribution from which samples can be drawn, the mapping of a new state, given the previous state and action, is learned directly. Such a model can be learned online during interaction with the real plant. During the *planning phase* the bi-RG-RRT algorithm is used to find a state space trajectory that minimizes the distance to a desired target state. The distance to the goal can be related to a cost-function that is minimized. As a result, it is also possible to define a reward function in this planning algorithm. Next, MACOP is employed to learn the corresponding actions iteratively and only based on the planned state space trajectory on the manifold of the learned simulation model of the plant. As a result, a policy is defined that can be applied on the real plant in the *execution phase*. This allows the forward model to collect new data, and the policy to adapt itself on the real plant to compensate for modeling inaccuracies. Given these similarities, the proposed control framework is closely related to simulation-based reinforcement learning approaches. For instance, the PILCO approach, which stands for "A Model-Based and Data-Efficient Approach to Policy Search", by Deisenroth and Rasmussen (2011), is in fact a simulation-based approach where Gaussian Processes (GPs) are used for their modeling capacity. Even when the number of data points is limited, they perform well. However, instead of sampling the state space, the gradient of the reward is calculated through a GP model of the plant's state transitions and the policy, which is also a GP. As a result, the policy is adjusted based on a simulation model of the plant, limiting the interaction time needed with the real plant. Thanks to the great modeling capability of a GP, PILCO performs really well.

6.7 Conclusions

In this chapter, the use of the IMA tracking controller is extended with a planning algorithm in order to make full control of an underactuated system possible. I used the cart pole experimental setup as an example for an underactuated system. The Rapidly-exploring Random Tree planning (RRT) algorithm, which is a sampling-based approach, was discussed. The Euclidean distance metric used in an RRT planner is, however, not suited for planning tasks under both kinematic and dynamic constraints. Therefore, I applied the Reachability Guided RRT algorithm that heuristically changes the Voronoi bias to more feasible sampling points. After a trajectory is planned in state space, MACOP is applied to convert this state space trajectory into the corresponding actions. MACOP distributes a set of IMA controllers according to the angular position of the pendulum. This allows each IMA controller to specialize its control in a sub-region of the state space. Additionally, the use of MACOP introduces feedback to the planned state space trajectory. As a result, the trajectory is stabilized and the target is reached even though there exist inaccuracies in the planned trajectory. These experiments were performed and demonstrated on an analytical model of the cart pole setup. However, the assumption made in this dissertation is that no prior model knowledge is available. Therefore, I demonstrated how this analytical model can be replaced by a learned forward model on which an RG-RRT planner can be applied. Instead of learning an accurate forward model prior to finding a trajectory that solves the task, I presented the concept of a simulation-based control framework that learns and explores the system dynamics simultaneously. However, the validation and evaluation of this concept remains to be done. Finally, I discuss the similarities between this simulation-based control framework and reinforcement learning.

7 Conclusions and Future Perspectives

In this final chapter, I give a brief summary of the work presented in this dissertation while highlighting my main contributions. Afterwards, a list of future research directions is provided which build upon the work described in this PhD thesis.

7.1 Summary and main achievements

During the last century, robots have been used increasingly to perform repetitive tasks more accurately and quickly than humans can. Their contribution in improving the efficiency of industrial processes, such as car manufacturing, have been proven to be beneficial. In an industrial setting these robots are surrounded by fences that prevent humans from entering the vicinity of the robot. Such robots can only operate in a deterministic environment where the unpredictable behavior of humans would result in dangerous situations where collisions are possible. In the last few decades, however, interest has emerged regarding the integration of robots in our daily lives. When integrating robots in a human friendly environment, more advanced motor/motion skills are needed. One approach to creating more advanced motion skills is that of imitation learning. Here, certain motions are demonstrated to the robot. The robot learns these motions in such a way that it can reproduce them. Furthermore, the robot should be able to generalize the motion to different situations of which no examples were shown.

In this work I have built a **Motion Pattern Generator (MPG)** using a Reservoir Computing approach called Echo State Networks (ESNs). This particular ESN is a dynamical system that can embed both periodic and discrete motion examples. By modifying/training this dynamical system appropriately the attractor landscape is shaped according to the motion examples. The shaping of this attractor landscape (i.e., manifold) also affects the regions around the demonstrations, yielding similar

motions for a different context than that of the examples. This generalization allows the motion generation to be more robust against perturbations; for instance, I have shown that when a robot manipulator is incorporated within the control loop it becomes possible to hold this manipulator, which pauses the motion generation as well. As soon as the manipulator is released the motion generation continues from where it was before the manipulator was perturbed. Although this allows a robot to handle perturbations, any delay in the control loop can cause high contact forces, yielding dangerous behavior. Consequently, engineers started to include force sensors in their robot designs. However, often the control loop that uses these sensors is too slow. Engineers and researchers have therefore advocated the use of soft materials and springs in their designs to construct so called compliant robots. As these materials react instantaneously to external forces, the initial contact can be compensated for by the structure of the robot while the remainder is compensated for by the slower control loop. However, these materials have the disadvantage that the corresponding robot becomes harder to control. Industrial manufacturing robots can rely on their sturdy structure to move according to an internal representation of the robot, also called a robot model. A robot with passive elements (e.g., soft materials, springs), on the other hand, is much harder and sometimes impossible to model. Therefore, the performance of the corresponding robot control will be insufficient.

To solve these problems, an adaptive control framework called **Inverse Modeling Adaptive (IMA) control framework** is proposed, that learns an inverse robot model by interacting with the robot. As the model becomes increasingly accurate the control performance improves as well. To create such models I also used ESNs; however, other modeling approaches are applicable as well. I have applied an ESN-based IMA controller to several control tasks, each with different dynamical behaviors, and demonstrated its widespread applicability. Furthermore, I have compared its performance to classical control approaches, which revealed its weaknesses and strengths. The IMA control framework is able to quickly learn the control within the regions of the desired target. Although it can generalize quite well to unseen regions, its performance is much better when the IMA controller has previously observed the data. I have argued that the IMA controller is a tracking controller that tries to move to the target directly. The IMA controller is not suitable for tasks where suboptimal movements are required before being able to reach a desired target. This is the case for underactuated systems such as the inverted pendulum.

Together with the developed Motion Pattern Generator, the IMA controller allows a compliant robot to be controlled in a manner that is familiar to humans. However, when the number of motions that are demonstrated increases it becomes hard to represent all of them in a single dynamical system. Therefore, it is preferable to spread the learning complexity over multiple MPGs. Such spreading of the complexity is something that can be observed in real biological systems; for in-

stance, experiments on frogs have shown that separate parts located in the lumbar cord are responsible for generating distinct force fields in the frog's motion generation. Simultaneous stimulation of these lumbar cord parts results in a superposition of the separate recorded force fields. This suggests that a mixing mechanism exists which allows a large collection of motions to be represented by a limited set of motions, called motion primitives. Furthermore, to increase the diversity, each motion primitive can be modulated slightly.

Inspired by this concept, I have investigated two control hierarchy types. The first control hierarchy assumes an MPG that is trained to generate a periodic motion. This MPG is used to directly control a quadruped robot leg. The actual motion is observed by an encoder, which allows for calculation of the current amplitude and the offset of the periodic motion. At a higher level the IMA controller attempts to learn the tracking of a desired amplitude or offset by modifying the MPG itself. In order to adjust the MPG, each neuron's bias was changed appropriately. Instead of adjusting the bias also other modulation approaches, such as adding an input, are applicable. Some perturbations of the robot leg cannot be compensated for by the passive elements of the leg. Such perturbations are the responsibility of the MPG, which is in direct control of the robot leg. However, slowly changing motion properties that are not compensated for by the MPG are modified by the IMA controller which is interacting at slower time scale. This difference in time scales among the different levels of the hierarchy is also biological plausible. Many aspects of the brain can be explained by a **hierarchy operating at different time scales**. Here, the higher level encodes slower contextual changes in the environment or body while at the lower level faster variations due to sensory processing are encoded. Some basic experiments have shown that the IMA controller can learn to control motion properties, such as amplitude and offset, indirectly. In these experiments I did not control both the amplitude and offset at the same time, which allowed the IMA controller to affect one property while controlling the other. This can be avoided by controlling both properties simultaneously.

The second control hierarchy, called **Modular Architecture with Control Primitives (MACOP)**, assumes that the primitives themselves are undefined. A single control primitive is represented by an untrained IMA controller. Before the control is initiated, only a high level description is given, stating that on average each primitive should contribute equally to the motion generation, while still allowing for each control primitive to be specialized in a part of the motion generation. As a result of this heuristic definition, the space in which the task is performed is tiled into subregions where a single primitive is more active than the others. I have applied MACOP on an inverse kinematic learning task where I have demonstrated that every control primitive is learning an inverse kinematic model. However, depending on a controller's contribution to a certain subregion, its accuracy will be better than other primitives. Furthermore, this spreading of the control task's com-

plexity over multiple controllers has shown to resolve redundancy problems. One can argue that by carefully choosing the space in which the tiling should take place, a single controller observes data that is in a way filtered to that particular region. This means that redundant solutions are represented by other controllers for a different region of the tiled space.

During my research, I have successfully applied MACOP to both a simulation model and a real implementation of a robot arm. Additionally, I have extended its use to **underactuated systems by incorporating a planning algorithm**. A planning algorithm defines a feasible path that needs to be followed to reach a certain target given an initial position. As mentioned before, an underactuated system such as an inverted pendulum cannot be controlled by an IMA controller when the swing-up is needed as well. However, by planning a trajectory that the IMA controller can follow the swing-up task should be solvable. The dynamics of an inverted pendulum are different depending on the position of the pendulum. Therefore, I have used MACOP to learn a set of controllers distributed over different possible ranges of pendulum positions. I have demonstrated that the task can indeed be solved when a planning algorithm is used in combination with MACOP. Furthermore, thanks to the feedback control of MACOP, inaccuracies in the planned trajectory are not critical for reaching the upward position. Although these experiments use an existing prior model of the pendulum, I have proposed a **simulation-based control framework** in which both the learning of a system model and the final control is learned at the same time. This simulation-based control framework allows underactuated systems to be controlled without any prior model knowledge. Finally, I have described the similarities of this simulation-based control framework with reinforcement learning.

Throughout this dissertation I have used ESNs as a default learning approach. Training all weights in a RNN will definitely yield a better model but requires much more time. In my work and in that of other researchers, ESNs have been shown to be well suited for control tasks where fast computations are needed, especially when using them in an online learning context. They have a number of parameters which need to tuned. In my experience, however, the most important parameters are the spectral radius, input-scaling, leak rate and network size. For the other parameters I often used default values. The main disadvantage of using ESNs is the large amount of data needed to learn a good model, especially when compared to non-linear regression approaches such a Gaussian Processes (GPs). Gaussian Processes, however, have the disadvantage that they become much slower when a large amount of data is available and they are notoriously slow to train. This is also the case for kernel-based approaches such as Support Vector Regression. In my opinion, choosing the best suited approach for the task at hand is very important because there is no single approach that does not have any disadvantages.

7.2 Future perspectives

During my four years of research into the field of robot control with Reservoir Computing, many new questions emerged when trying to solve a particular problem. As with any research many of these questions remain. In this last section I will give an overview of some of these questions, organized according to the structure of this dissertation.

Reservoir Computing

Throughout my work I have used Reservoir Computing to approximate the dynamics of a dynamical system. When using them in control applications it is necessary to provide rigorous proofs of their performance and stability, as well as the constraints under which asymptotic stability is guaranteed. It is clear that a possibly unstable controller cannot be used as an automatic pilot in an airplane. However, as I have mentioned in Section 4.2, Recurrent Neural Networks (RNNs) are notoriously difficult to analyze, especially in the non-autonomous case. Although proofs based on linearized network dynamics exist, there is still a need for global asymptotic stability proofs for the full non-linear dynamics.

Another interesting research direction is that of Morphological Computing. This research field investigates how the dynamics of the body can be exploited to perform computations. For instance, within a control application it should be possible to move a part from the control complexity to the body of the robot, simplifying the actual control. When the structure of the robot has dynamic properties it behaves as a dynamical system, similar to RNN behavior. As a result, a mapping between the robot's structure and a corresponding RNN that was trained to behave as controller might exist. This mapping could allow the robot's body to take over a part of the computations needed to actually control itself and to adapt to a changing environment. Such morphological computing is a very interesting research direction and poses many unsolved questions.

Motion pattern generator

In Chapter 3 the design of a Motion Pattern Generator (MPG) was discussed. This MPG allows the embedding of both periodic and discrete motions into a single generator. During the switching from one motion to the other, a transient behavior is observed. An interesting research direction is to investigate how these transient behaviors can be adjusted. It is possible to show demonstrations of a desired transient motion during the training phase. However, when only demonstrations of each separate motion are shown and not the transient motion between them, these transients

might not be desired. It might be possible to modulate an MPG's dynamics after training in order that a preferred transient behavior is achieved. If this is possible, any undesired behavior can be modulated allowing that only stable and desired motions trajectories are generated.

Adaptive control framework

The IMA controller, proposed in Chapter 4, has been applied on a wide variety of tasks. Under the hood it uses a high dimensional feature representation, from which an inverse model is build during the interaction with the plant/robot that needs to be controlled. Many learning approaches are based on such high dimensional representation and can therefore be used. However, in this work I only used ESNs for the implementation of an IMA controller. ESNs are in general easy to use, fast to train and execute. Furthermore, ESNs can benefit from parallelization mechanisms. Their main disadvantage is the amount of data needed to achieve a good model. Investigating multiple learning approaches and comparing their performance is something which remains to be done.

In Chapter 4, I mentioned that it is important to tweak the δ parameter in order for the plant dynamics to be modeled. At the moment this parameter needs to be optimized by hand or by applying a grid search; however, it should be possible to calculate this δ based on observation of the plant/robot dynamics. One could for instance apply a step function as an input, observe the plant/robot response to such impulses and derive the most appropriate parameter from this. In most cases this parameter will correspond to the delay in the response of the plant to a sudden change in input. Furthermore, it might be possible to change this dynamically so that this parameter can be adjusted according to the changes in dynamic behavior of the plant/robot. This will allow an IMA controller to learn the control of even more complex systems.

Control hierarchies

In Section 5.1, I presented a neural hierarchy of time scales and demonstrated its plausibility using a very simple experiment. During my four years of research I also investigated how multiple primitives, each generated by a single MPG, can be combined to achieve more advanced motion control. As illustrated in panel **A** of Figure 7.1, I have tried to use an IMA controller in such a way that it learns to modulate a set of very basic motor primitives (up, down, left, right). Results of these experiments can be found in the Appendix A.2. Although I was successful in controlling a very basic omni-wheeled mobile robot in such a way that it followed a desired trajectory, I was unsuccessful in achieving the same results in a more advanced setup. For example, I tried to control the joint angles of a robot arm

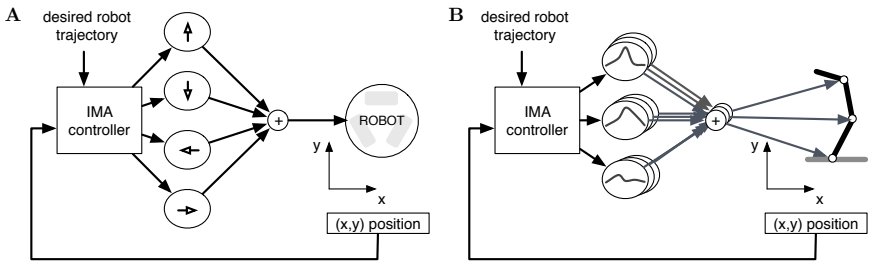


Figure 7.1: Left configuration illustrates an experiment where simple motion primitives are combined so that an omni-directional robot follows a target trajectory. Right configuration is a more advanced experiment where the joint positions of a planar robot are controlled by the mixing of a set of Gaussian shaped primitives.

with a set of Gaussian shaped primitives, for which the IMA controller learns the gating between them (panel **B** of Figure 7.1). Unfortunately, these experiments were unsuccessful and will require a more thorough investigation.

Section 5.2 describes a modular architecture with control primitives (MACOP) that defines a heuristic under which motion control primitives are learned without supervision. Although this heuristic rule allows the task to be solved successfully, there may be alternatives which have other advantages. Furthermore, it would be interesting to investigate the biological plausibility of this unsupervised emergence of control primitives.

I have applied MACOP on two different robots and in both cases the inverse kinematics were learned. However, it remains to be seen how other tasks with other robots can benefit from using MACOP.

Motion planning and control

Chapter 6 describes the extension of the IMA controller with a RRT planning algorithm allowing that underactuated systems can be controlled. I showed how MACOP can be applied to learn the necessary torques in order to follow the planned trajectory and how this trajectory is stabilized. I demonstrated that every part of the presented simulation-based control framework works. However, a more thorough investigation and evaluation of this framework is required. Furthermore, it would be interesting to compare the performance of this framework to that of similar simulation-based techniques.

7.3 Epilogue

In this dissertation a set of Reservoir Computing based approaches were presented that can be used in the context of adaptive control. Not only have I demonstrated that they are applicable on a large number of simulation tasks, each with their own interesting properties, but also on some real compliant robot tasks. The presented control approach can be applied on tasks for which it is hard to model certain physical properties, e.g., compliant robots with elastic materials. This does not mean that model-based controllers are rendered useless. On the contrary, model information should be used when it is available. Furthermore, they can be used complementary where the model inaccuracies of a model-based controller are compensated by the presented control approach. Although compliance was introduced to improve the human-robot interaction, safety was not the motivation of this thesis. Whether the presented approaches actually improve the interaction safety, still needs to be investigated.

A Appendix

A.1 Kinematic model 3R manipulator

For a three degree of freedom (DOF) planar redundant manipulator, the forward kinematic equations are formulated as follows:

$$\begin{pmatrix} x \\ y \end{pmatrix}^T = \begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix}^T \begin{pmatrix} \cos(\theta_1) & \sin(\theta_1) \\ \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) \\ \cos(\theta_1+\theta_2+\theta_3) & \sin(\theta_1+\theta_2+\theta_3) \end{pmatrix}$$

$$\phi = \theta_1 + \theta_2 + \theta_3, \tag{A.1}$$

where the location in a 2D Cartesian plane is defined by x and y. ϕ indicates the angle of the end-effector with respect to the robot's base. As shown in Figure 3.2, l_1, l_2 and l_3 define the first, second and third link length respectively while α_1, α_2 and α_3 denote their associated angles. The robotic manipulator used has the following dimensions: $l_1 = l_2 = 9.15$ cm and $l_3 = 3.5$ cm.

The inverse kinematics necessary for actual manipulator control are calculated as follows:

$$\theta_1 = \arctan\left(\frac{y_0}{x_0}\right) + \arccos\left(\frac{l_1^2 - l_2^2 + x_0^2 + y_0^2}{2l_1\sqrt{x_0^2+y_0^2}}\right) \tag{A.2}$$

$$\theta_2 = \arccos\left(\frac{x_0^2 + y_0^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \tag{A.3}$$

$$\theta_3 = \phi - (\theta_1 + \theta_2), \tag{A.4}$$

with $x_0 = x - l_3\cos(\phi)$ and $y_0 = y - l_3\sin(\phi)$.

Table A.1: Network parameters of the sequencing task

Parameter	Value	Parameter	Value
N	400 neurons	f_i^r	0.1
ρ	1	f_b^r	0.1
δ	2	γ	1

A.2 Sequencing of motion primitives

In this little experiment an automated manner of finding a correct sequence of motion primitives is investigated. An omni-directional robot is directly driven by a combination of 4 possible motion primitives: drive 1 cm forward, drive 1 cm backward, drive 1 cm to the left or drive 1 cm to the right. In order to learn the correct sequence of motion primitives, to follow a desired trajectory, I used an IMA controller. As shown in panel **A** of Figure 7.1, this IMA controller has 4 outputs, one for each primitive. If a single output obtains a value above 0, the corresponding primitive will be activated. The IMA controller receives the actual two dimensional trajectory of the robot's position as feedback. The network parameters of the ESN-based IMA controller are shown in Table A.1.

As shown in Figure A.1, the robot starts at position $(0,0)$ and needs to follow a circular trajectory. After an initial exploration, the robot starts to track the target trajectory by combining primitive motions. The top plot illustrates this initial exploration from blue to green. The bottom plot illustrates the on and off switching of the primitives. One can notice that the IMA controller has learned to combine for example, 'up' and 'down' to stay at the same y position. Another (redundant) solution to keep the y position fixed would be to disable both the 'up' and 'down' primitive. The same can be said about the other two primitives.

Figure A.2 demonstrates the results after 3000 time steps. It is clear that the IMA controller has learned to track the desired trajectory even better. Each primitive can take the robot 1 cm further from its current position. Such steps are sometimes to fast, compared to the target position. Consequently, in the top left part of the circle the robot moves forward an backward, in order to keep track of the desired trajectory.

I have tried to extend this simple experiment to a more complicated setup (e.g., panel **B** of Figure 7.1). However, I was unable to get good results. In order to achieve good and robust results, a more extensive investigation and evaluation is needed.

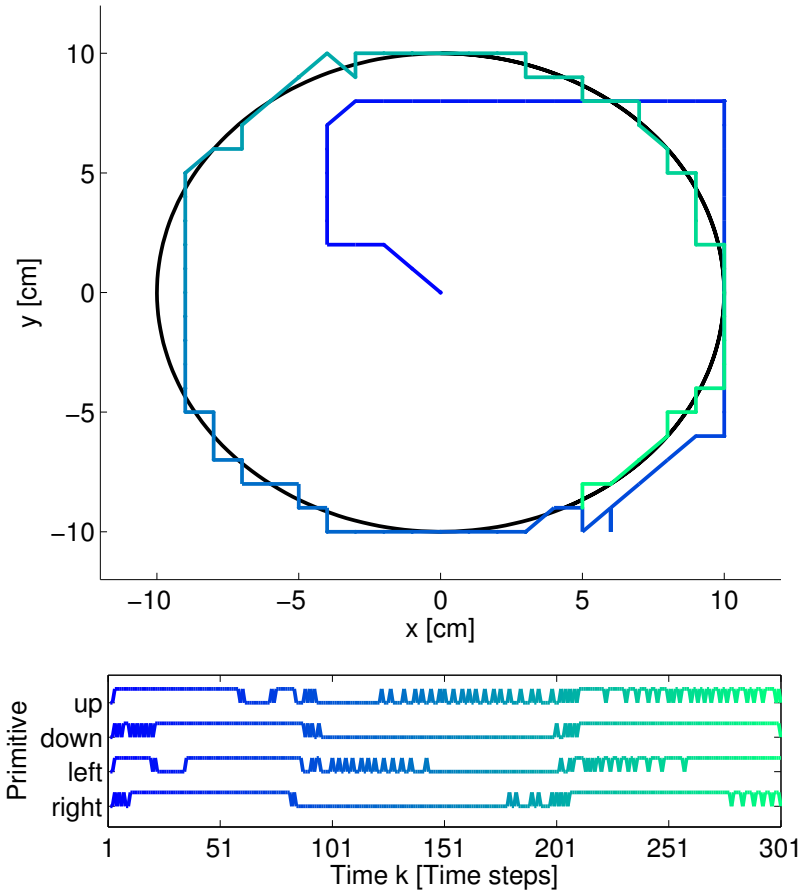


Figure A.1: The top plot shows the robot's movement compared to its target trajectory. The color of the trajectory gives an indication of how the trajectory progresses, from blue to green. The bottom plot illustrates when a particular primitive is activated.

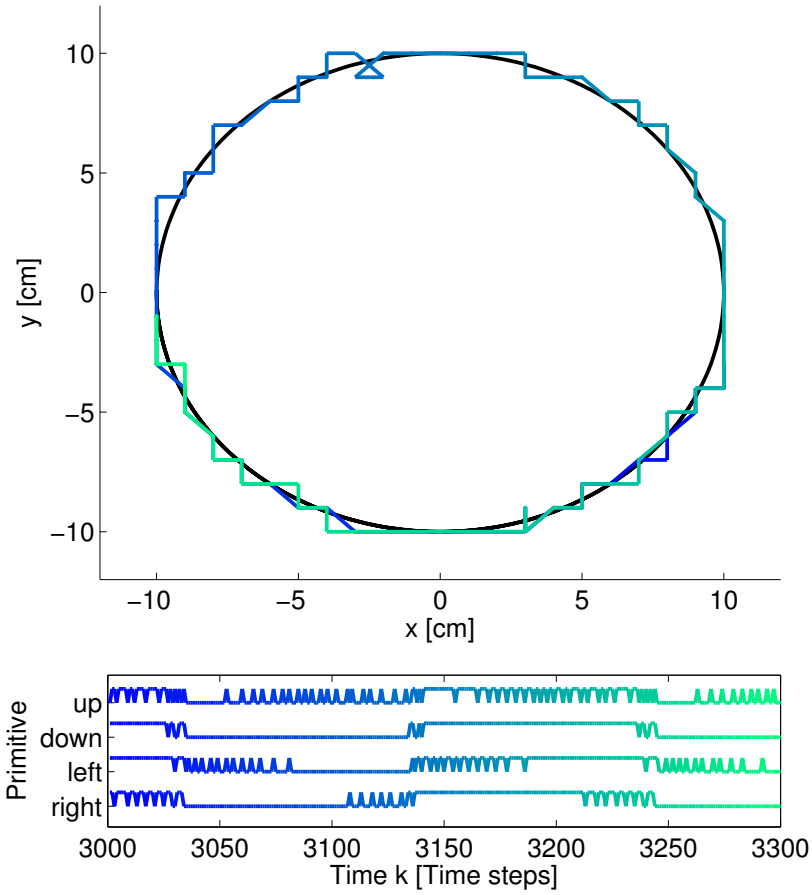


Figure A.2: The top plot demonstrates the robot's movement compared to its target trajectory. The color of the trajectory gives an indication of how the trajectory progresses, from blue to green. The bottom plot illustrates when a particular primitive is activated.

Bibliography

Acheson, D. J. (1990). *Elementary fluid dynamics*. Oxford University Press.

Ackermann, J., Bartlett, A., Kaesbauer, D., Sienel, W., and Steinhauser, R. (1993). *Robust control*. Springer.

Agachi, P. S., Nagy, Z. K., Cristea, M. V., and Imre-Lucaci, Á. (2007). *Model based control*. John Wiley & Sons.

Ahmed, M. R. (2011). *Compliance Control of Robot Manipulator for Safe Physical Human Robot Interaction*. PhD thesis, Orebro University.

Ajallooeian, M., Gay, S. b., Ijspeert, A., Khansari-Zadeh, M., Kim, S., Billard, A., Rü ckert, E., Neumann, G., Waegeman, T., wyffels, F., Schrauwen, B., Lemme, A., Reinhart, F., Rolf, M., Steil, J., Carbajal, J. P., Sumioka, H., Zhao, Q., and Kuppuswamy, N. (2010). Comparative evaluation of approaches in t.4.1-4.3 and working definition of adaptive module.

Albert, A. (1972). *Regression and the Moore-Penrose pseudoinverse*, volume 3. Academic Press New York.

Albu-Schaffer, A., Eiberger, O., Grebenstein, M., Haddadin, S., Ott, C., Wimbock, T., Wolf, S., and Hirzinger, G. (2008). Soft robotics. *Robotics & Automation Magazine, IEEE*, 15(3):20–30.

Albu-Schaffer, A., Ott, C., and Hirzinger, G. (2007). A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *The International Journal of Robotics Research*, 26(1):23–39.

Antonelo, E. (2011). *Reservoir computing architectures for modeling robot navigation systems*. PhD thesis, Ghent University.

Arbib, M. A. (2003). *The handbook of brain theory and neural networks*. The MIT press.

Asada, H. and Youcef-Toumi, K. (1987). *Direct-drive robots: theory and practice*. MIT press.

Åström, K. J. and Wittenmark, B. (2008). *Adaptive control*. Courier Dover Publications.

Atiya, A. and Parlos, A. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709.

Avi-Itzhak, H. I., Diep, T. A., and Garland, H. (1995). High accuracy optical character recognition using neural networks with centroid dithering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(2):218–224.

Barabanov, N. and Prokhorov, D. (2003). A new method for stability analysis of nonlinear discrete-time systems. *IEEE Transactions on Automatic Control*, 48(12):2250–2255.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bernstein, N. (1967). The problem of interrelation of coordination and localization. *The coordination and regulation of movements*, 80(4):15–59.

Bertsekas, D. P. and Tsitsiklis, J. N. (1995). Neuro-dynamic programming: An overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE.

Bicchi, A. and Tonietti, G. (2004). Fast and soft-arm tactics. *IEEE Robotics and Automation Magazine*, page 22.

Bicchi, A., Tonietti, G., Bavaro, M., and Piccigallo, M. (2005). Variable stiffness actuators for fast and safe motion control. In *Robotics Research*, pages 527–536. Springer.

Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*, volume 1. springer New York.

Bizzi, E., Mussa-Ivaldi, F., Giszter, S., et al. (1991). Computations underlying the execution of movement: a biological perspective. *Science*, 253(5017):287–291.

Boyd, S. and Chua, L. (1985). Fading memory and the problem of approximating nonlinear operators with volterra series. *IEEE Transactions on Circuits and Systems*, 32(11):1150–1161.

Brosilow, C. and Joseph, B. (2002). *Techniques of model based control*. Prentice Hall Professional.

Büsing, L., Schrauwen, B., and Legenstein, R. (2010). Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons. *Neural computation*, 22(5):1272–1311.

Buteneers, P. (2012). *Detection of epileptic seizures: the reservoir computing approach*. PhD thesis, Ghent University.

Buteneers, P., Verstraeten, D., Nieuwenhuyse, B. V., Stroobandt, D., Raedt, R., Vonck, K., Boon, P., and Schrauwen, B. (2012). Real-time detection of epileptic seizures in animal models using reservoir computing.

Caluwaerts, K., D'Haene, M., Verstraeten, D., and Schrauwen, B. (2013a). Locomotion without a brain: Physical reservoir computing in tensegrity structures. *Artificial life*, 19(1):35–66.

Caluwaerts, K. and Schrauwen, B. (2011). The body as a reservoir : locomotion and sensing with linear feedback. In *Proceedings of the 2nd International Conference on Morphological Computation*.

Caluwaerts, K., wyffels, F., Dieleman, S., and Schrauwen, B. (2013b). The spectral radius remains a valid indicator of the echo state property for large reservoirs. In *International Joint Conference on Neural Networks (IJCNN-2013)*.

Cessac, B. (2010). A view of neural networks as dynamical systems. *International Journal of Bifurcation and Chaos*, 20(06):1585–1629.

Cheng, G. S.-X. (2000). Model-free adaptive process control. US Patent 6,055,524.

Cheung, V., d'Avella, A., Tresch, M., and Bizzi, E. (2005). Central and sensory contributions to the activation and organization of muscle synergies during natural motor behaviors. *The Journal of neuroscience*, 25(27):6419–6434.

Chew, C.-M., Hong, G.-S., and Zhou, W. (2004). Series damper actuator: a novel force-torque control actuator. In *2004 4th IEEE/RAS International Conference on Humanoid Robots*, volume 2, pages 533–546. IEEE.

Chow, T. and Fang, Y. (1998). A recurrent neural-network-based real-time learning control strategy applying to nonlinear systems with unknown dynamics. *IEEE Transactions on Industrial Electronics*, 45(1):151–161.

Cohen, A. H. and Wallen, P. (1980). The neuronal correlate of locomotion in fish. *Experimental brain research*, 41(1):11–18.

Crespi, A., Karakasiliotis, K., Guignard, A., and Ijspeert, A. J. (2013). Salamandra robotica ii: an amphibious robot to study salamander-like swimming and walking gaits.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

Daerden, F. and Lefeber, D. (2001). The concept and design of pleated pneumatic artificial muscles. *International Journal of Fluid Power*, 2(3):41–50.

Dambre, J., Verstraeten, D., Schrauwen, B., and Massar, S. (2012). Information processing capacity of dynamical systems. *Scientific reports*, 2.

d'Avella, A. and Bizzi, E. (2005). Shared and specific muscle synergies in natural motor behaviors. *Proceedings of the National Academy of Sciences of the United States of America*, 102(8):3076–3081.

d'Avella, A., Saltiel, P., and Bizzi, E. (2003). Combinations of muscle synergies in the construction of a natural motor behavior. *Nature neuroscience*, 6(3):300–308.

De Keyser, R. (2003). Model Based Predictive Control, Invited Chapter in UNESCO Encyclopaedia of Life Support Systems (EoLSS).

De Luca, A., Albu-Schaffer, A., Haddadin, S., and Hirzinger, G. (2006). Collision detection and safe reaction with the dlr-iii lightweight manipulator arm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1623–1630. IEEE.

De Schutter, J., Bruyninckx, H., Zhu, W.-H., and Spong, M. W. (1998). Force control: a bird's eye view. In *Control Problems in Robotics and Automation*, pages 1–17. Springer.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 465–472.

Dhamala, M., Jirsa, V. K., and Ding, M. (2004). Enhancement of neural synchrony by time delay. *Physical review letters*, 92(7):074104.

Doeringer, J. and Hogan, N. (1998). Intermittency in preplanned elbow movements persists in the absence of visual feedback. *Journal of neurophysiology*, 80(4):1787–1799.

Dominey, P. F. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological cybernetics*, 73(3):265–274.

Dormand, J. and Prince, P. (1980). A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26.

Dossche, M. (2012). Een vergelijkende studie over de controle van de geïnverteerde pendel.

Drossel, B. (2008). 3 random boolean networks. *Reviews of nonlinear dynamics and complexity*, 1:69.

Fernando, C. and Sojakka, S. (2003). Pattern recognition in a bucket. In *Advances in Artificial Life*, pages 588–597. Springer.

Flash, T. and Hochner, B. (2005). Motor primitives in vertebrates and invertebrates. *Current opinion in neurobiology*, 15(6):660–666.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control, and Dynamics*, 25(1):116–129.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Gálvez-Carrillo, M., De Keyser, R., and Ionescu, C. (2009). Nonlinear predictive control with dead-time compensator: Application to a solar power plant. *Solar Energy*, 83(5):743–752.

Ge, S., Yang, C., and Lee, T. (2008). Adaptive predictive control using neural network for a class of pure-feedback systems in discrete time. *IEEE Transactions on Neural Networks*, 19(9):1599–1614.

Gennarelli, T. A. and Wodzin, E. (2006). Ais 2005: A contemporary injury scale. *Injury*, 37(12):1083–1091.

Glassman, E. and Tedrake, R. (2010). A quadratic regulator-based heuristic for rapidly exploring state space. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028. IEEE.

Graves, A., Liwicki, M., Fernández, S., Bertolami, R., Bunke, H., and Schmidhuber, J. (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868.

Haddadin, S. (2014). Considerations for new robot standards. In *Towards Safe Robots*, pages 317–336. Springer.

Hafner, R. and Riedmiller, M. (2011). Reinforcement learning in feedback control. *Machine learning*, 84(1-2):137–169.

Ham, R., Sugar, T., Vanderborght, B., Hollander, K., and Lefeber, D. (2009). Compliant actuator designs. *Robotics & Automation Magazine, IEEE*, 16(3):81–94.

Han, J. (2009). From pid to active disturbance rejection control. *IEEE Transactions on Industrial Electronics*, 56(3):900–906.

Harper, C. and Virk, G. (2010). Towards the development of international safety standards for human robot interaction. *International Journal of Social Robotics*, 2(3):229–234.

Hart, C. and Giszter, S. (2004). Modular premotor drives and unit bursts as primitives for frog motor behaviors. *The Journal of neuroscience*, 24(22):5269–5282.

Haruno, M., Wolpert, D., and Kawato, M. (2001). Mosaic model for sensorimotor learning and control. *Neural computation*, 13(10):2201–2220.

Hauser, H., Ijspeert, A., Füchslin, R., Pfeifer, R., and Maass, W. (2012). Towards a theoretical foundation for morphological computation with compliant bodies. *Biological Cybernetics*. in press.

Hermans, M. (2012). *Expanding the theoretical framework of reservoir computing*. PhD thesis, Ghent University.

Hermans, M. and Schrauwen, B. (2011). Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Horn, J., Nafpliotis, N., and Goldberg, D. (1994). A niched pareto genetic algorithm for multiobjective optimization. In *Proc. of the First IEEE Conference on Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence.*, pages 82–87. IEEE.

Hung, L. and Chung, H. (2007). Decoupled control using neural network-based sliding-mode controller for nonlinear systems. *Expert Systems with Applications*, 32(4):1168–1182.

Hurst, J. W., Chestnutt, J. E., and Rizzi, A. A. (2010). The actuator with mechanically adjustable series compliance. *IEEE Transactions on Robotics*, 26(4):597–606.

Ifeachor, E. C. and Jervis, B. W. (2002). *Digital signal processing: a practical approach*. Pearson Education.

Ijspeert, A., Crespi, A., Ryczko, D., and Cabelguen, J. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, 315(5817):1416–1420.

Ijspeert, A. J. (2008). Central pattern generators for locomotion control in animals and robots: a review. *Neural Networks*, 21(4):642–653.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1523–1530.

Ilin, A., Valpola, H., and Oja, E. (2004). Nonlinear dynamical factor analysis for state change detection. *IEEE Transactions on Neural Networks*, 15(3):559–575.

Ismail, M. and Principe, J. (1996). Equivalence between rls algorithms and the ridge regression technique. In *Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers, 1996.*, pages 1083–1087. IEEE.

Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks. Technical report, Technical Report GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. (2002). A method for supervised teaching of a recurrent artificial neural network. Patent Application WO 2002/031764 A2.

Jaeger, H. (2010). Reservoir self-control for achieving invariance against slow input distortions. Technical report, Technical report, Jacobs University.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78.

Jaeger, H., Lukosevicius, M., Popovici, D., and Siewert, U. (2007). Optimization and applications of echo state networks with leaky-integrator neurons. *Neural Networks*, 20(3):335–352.

Jalali, A., Piltan, F., Gavahian, A., Jalali, M., et al. (2013). Model-free adaptive fuzzy sliding mode controller optimized by particle swarm for robot manipulator. *International Journal of Information Engineering and Electronic Business (IJIEEB)*, 5(1):68.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., et al. (2000). *Principles of neural science*, volume 4. McGraw-Hill New York.

Kargo, W. and Giszter, S. (2000). Rapid correction of aimed movements by summation of force-field primitives. *The Journal of Neuroscience*, 20(1):409–426.

Karoń, I. (2012). Modified model-free adaptive controller for a nonlinear rotor system. In *Artificial Intelligence and Soft Computing*, pages 458–465. Springer.

Kawato, M., Furukawa, K., and Suzuki, R. (1987). A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57(3):169–185.

Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957.

Khansari-Zadeh, S. M. and Billard, A. (2012). A dynamical system approach to realtime obstacle avoidance. *Autonomous Robots*, 32(4):433–454.

Khansari-Zadeh, S. M., Seungsu, K., Rückert, E. A., Waegeman, T., Lemme, A., Reinhart, F., Ajallooeian, M., and Gay, S. (2010). D 4.1, reaching benchmark (v1) results.

Kiebel, S. J., Daunizeau, J., and Friston, K. J. (2008). A hierarchy of time-scales and the brain. *PLoS computational biology*, 4(11):e1000209.

Kim, B. S. and Calise, A. J. (1997). Nonlinear flight control using neural networks. *Journal of Guidance, Control, and Dynamics*, 20(1):26–33.

Kim, M., Yun, S., and Kang, S. (2004). Safe design and validation control of a manipulator with passive compliant joints. In *2nd Int. Conf. on Autonomous robots and agents, Palmerston North, New Zealand*, pages 13–15.

Kohonen, T. (1998). The self-organizing map. *Neurocomputing*, 21(1):1–6.

Krose, B. J., van der Korst, M. J., and Groen, F. (1990). Learning strategies for a vision based neural controller for a robot arm. In *1990. Proceedings of the IEEE International Workshop on Intelligent Motion Control*, volume 1, pages 199–203. IEEE.

Kuffner Jr, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA'00*, volume 2, pages 995–1001. IEEE.

Kundu, P. and Cohen, I. (2008). Fluid mechanics. 2004.

Kwakernaak, H. and Sivan, R. (1972). *Linear optimal control systems*, volume 172. Wiley-Interscience New York.

Lamb, J. S. and Roberts, J. A. (1998). Time-reversal symmetry in dynamical systems: a survey. *Physica D: Nonlinear Phenomena*, 112(1):1–39.

Laurin-Kovitz, K. F., Colgate, J. E., and Carnes, S. D. (1991). Design of components for programmable passive impedance. In *1991 IEEE International Conference on Robotics and Automation, Proceedings., 1991*, pages 1476–1481. IEEE.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LaValle, S. M. and Kuffner Jr, J. J. (2000). Rapidly-exploring random trees: Progress and prospects.

Lefebvre, T., Xiao, J., Bruyninckx, H., and De Gersem, G. (2005). Active compliant motion: a survey. *Advanced Robotics*, 19(5):479–499.

Levin, A. and Narendra, K. (1996). Control of nonlinear dynamical systems using neural networks. ii. observability, identification, and control. *IEEE Transactions on Neural Networks*, 7(1):30–42.

Lewis, F. L., Vrabie, D., and Syrmos, V. L. (2012). *Optimal control*. Wiley. com.

Li, J. and Jaeger, H. (2011). Minimal energy control of an esn pattern generator. *Technical report, Jacobs University*.

Lorenz, E. (1963). Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141.

Lukosevicius, M. and Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149.

Maass, W. (2010). Liquid state machines: Motivation, theory, and applications. In Cooper, B. and Sorbi, A., editors, *Computability in Context: Computation and Logic in the Real World*, pages 275–296. Imperial College Press.

Maass, W., Natschläger, T., and Markram, H. (2002). Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560.

Mackey, M. and Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science New Series*, 197(4300):287–289.

Maeda, G. J., Singh, S. P., and Durrant-Whyte, H. (2010). Feedback motion planning approach for nonlinear control using gain scheduled rrts. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),*, pages 119–126. IEEE.

Maki, Y. and Loparo, K. A. (1997). A neural-network approach to fault detection and diagnosis in industrial processes. *IEEE Transactions on Control Systems Technology*, 5(6):529–541.

Margheri, L., Laschi, C., and Mazzolai, B. (2012). Soft robotic arm inspired by the octopus: I. from biological functions to artificial requirements. *Bioinspiration & Biomimetics*, 7(2):025004.

Marzban, C. and Stumpf, G. J. (1996). A neural network for tornado prediction based on doppler radar-derived attributes. *Journal of Applied Meteorology*, 35(5):617–626.

Massie, T. H. and Salisbury, J. K. (1994). The phantom haptic interface: A device for probing virtual objects. In *Proceedings of the ASME winter annual meeting, symposium on haptic interfaces for virtual environment and teleoperator systems*, volume 55, pages 295–300. IOS Press.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

Messner, B. and Tilbury, D. (1996). Digital Control Tutorial by University of Michigan and Carnegie Mellon.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Mitchell, T. M. (2006). *The discipline of machine learning*. Carnegie Mellon University, School of Computer Science, Machine Learning Department.

Morrell, J. B. and Salisbury, J. K. (1998). Parallel-coupled micro-macro actuators. *The International Journal of Robotics Research*, 17(7):773–791.

Muelling, K., Kober, J., and Peters, J. (2010). Learning table tennis with a mixture of motor primitives. In *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pages 411–416. IEEE.

Murphy, M. P., Saunders, A., Moreira, C., Rizzi, A. A., and Raibert, M. (2011). The littledog robot. *The International Journal of Robotics Research*, 30(2):145–149.

Mussa-Ivaldi, F. and Bizzi, E. (2000). Motor learning through the combination of primitives. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 355(1404):1755–1769.

Mussa-Ivaldi, F., Giszter, S., and Bizzi, E. (1994). Linear combinations of primitives in vertebrate motor control. *Proceedings of the National Academy of Sciences*, 91(16):7534–7538.

Nakajima, K., Hauser, H., Kang, R., Guglielmino, E., Caldwell, D. G., and Pfeifer, R. (2013). A soft body as a reservoir: Case studies in a dynamic model of octopus-inspired soft robotic arm. *Frontiers in Computational Neuroscience*, 7:91.

Narendra, K. and Parthasarathy, K. (1989). Adaptive identification and control of dynamical systems using neural networks. In *Proceedings of the 28th IEEE Conference on Decision and Control*, pages 1737–1738.

Needham, J. (1976). *Science and Civilisation in China: Historical Survey, from Cinnabar Elixirs to Synthetic Insulin. Vol. 5, Chemistry and chemical technology; Pt. 3, Spagyrical discovery and invention*, volume 3. Cambridge University Press.

Nguyen, D. and Widrow, B. (1989). The truck backer-upper: an example of self-learning in neuralnetworks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 357–363.

Nikolić, D., Haeusler, S., Singer, W., and Maass, W. (2006). Temporal dynamics of information content carried by neurons in the primary visual cortex. In *Advances in neural information processing systems*, pages 1041–1048.

Nori, F. and Frezza, R. (2004). Biologically inspired control of a kinematic chain using the superposition of motion primitives. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 1075–1080. IEEE.

Odhner, L. U., Jentoft, L. P., Claffee, M. R., Corson, N., Tenzer, Y., Ma, R. R., Buehler, M., Kohout, R., Howe, R. D., and Dollar, A. M. (2013). A compliant, underactuated hand for robust manipulation. *arXiv preprint arXiv:1301.4394*.

Oyama, E., Agah, A., MacDorman, K., Maeda, T., and Tachi, S. (2001). A modular neural network architecture for inverse kinematics model learning. *Neurocomputing*, 38:797–805.

Ozturk, M. C. (2007). *Dynamical Computation with Echo State Networks*. PhD thesis, University of Florida.

Pan, Y. and Wang, J. (2011). Model predictive control of unknown nonlinear dynamical systems based on recurrent neural networks. *IEEE Transactions on Industrial Electronics*, (99):1–1.

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

Pfeifer, R. and Bongard, J. (2007). *How the body shapes the way we think: a new view of intelligence*. MIT press.

Pfeifer, R. and Iida, F. (2005). Morphological computation: Connecting body, brain and environment. *Japanese Scientific Monthly*, 58(2):48–54.

Plackett, R. L. (1950). Some theorems in least squares. *Biometrika*, 37(1/2):149–157.

Potkonjak, V., Svetozarevic, B., Jovanovic, K., and Holland, O. (2010). Biologically-inspired control of a compliant anthropomimetic robot. In *The 15th IASTED International Conference on Robotics and Applications*, pages 182–189.

Pratt, G. A. and Williamson, M. M. (1995). Series elastic actuators. In *Proceedings of 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots'*, volume 1, pages 399–406. IEEE.

Prokhorov, D. (2007). Training recurrent neurocontrollers for real-time applications. *IEEE Transactions on Neural Networks*, 18(4):1003–1015.

Provenzale, A., Smith, L. A., Vio, R., and Murante, G. (1992). Distinguishing between low-dimensional dynamics and randomness in measured time series. *Physica D: nonlinear phenomena*, 58(1):31–49.

Reinhart, R. F. (2011). Reservoir computing with output feedback. *KI-Künstliche Intelligenz*, pages 1–2.

Reinhart, R. F. and Steil, J. J. (2009). Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub. In *9th IEEE-RAS International Conference on Humanoid Robots, 2009. Humanoids 2009.*, pages 323–330. IEEE.

Richard, J. (2003). Time-delay systems: an overview of some recent advances and open problems. *Automatica*, 39(10):1667–1694.

Riedmiller, M. (2005). Neural reinforcement learning to swing-up and balance a real pole. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3191–3196. IEEE.

Rigatos, G. G. (2009). Model-based and model-free control of flexible-link robots: a comparison between representative methods. *Applied Mathematical Modelling*, 33(10):3906–3925.

Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, DTIC Document.

Rossignol, S. (2000). Locomotion and its recovery after spinal injury. *Current opinion in neurobiology*, 10(6):708–716.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (2002). Learning representations by back-propagating errors. *Cognitive modeling*, 1:213.

Samuel, A. L. (2000). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226.

Scarfogliero, U., Stefanini, C., and Dario, P. (2009). The use of compliant joints and elastic energy storage in bio-inspired legged robots. *Mechanism and Machine Theory*, 44(3):580–590.

Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. *Adaptive motion of animals and machines*, page 261.

Schaal, S., Ijspeert, A., Billard, A., Schaal, S., Ijspeert, A., and Billard, A. (2003). Computational approaches to motor learning by imitation. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 358(1431):537–547.

Schaal, S., Kotosaka, S., and Sternad, D. (2000). Nonlinear dynamical systems as movement primitives. In *IEEE International Conference on Humanoid Robotics*.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. (2005). Learning movement primitives. *Robotics Research*, pages 561–572.

Schiavi, R., Bicchi, A., and Flacco, F. (2009). Integration of active and passive compliance control for safe human-robot coexistence. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 259–264. IEEE.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Schrauwen, B., Wardermann, M., Verstraeten, D., Steil, J. J., and Stroobandt, D. (2008). Improving reservoirs using intrinsic plasticity. *Neurocomputing*, 71(7):1159–1171.

Schürmann, F., Meier, K., and Schemmel, J. (2004). Edge of chaos computation in mixed-mode vlsi-a hard liquid. In *Advances in Neural Information Processing Systems*, pages 1201–1208.

Semini, C., Tsagarakis, N. G., Guglielmino, E., Focchi, M., Cannella, F., and Caldwell, D. G. (2011). Design of hyq: a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 225(6):831–849.

Shepherd, G. M. et al. (2004). *The synaptic organization of the brain*, volume 4. Oxford University Press Oxford.

Shkolnik, A. and Tedrake, R. (2009). Path planning in 1000+ dimensions using a task-space voronoi bias. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 2061–2067. IEEE.

Shkolnik, A., Walter, M., and Tedrake, R. (2009). Reachability-guided sampling for planning under differential constraints. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 2859–2865. IEEE.

Siciliano, B. and Khatib, O. (2008). *Springer handbook of robotics*. Springer.

Skowronski, M. D. and Harris, J. G. (2004). Exploiting independent filter bandwidth of human factor cepstral coefficients in automatic speech recognition. *Journal of the Acoustical Society of America*, 116(3):1774–1780.

Sontag, E. (1998). *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer-Verlag.

Spooner, J. and Passino, K. (1996). Stable adaptive control using fuzzy systems and neural networks. *IEEE Transaction on Fuzzy Systems*, 4(3):339–359.

Sproewitz, A., Ijspeert, A., et al. (2011). Oncilla robot: a light-weight bioinspired quadruped robot for fast locomotion in rough terrain. In *Proc. 5th International Symposium on Adaptive Motion on Animals and Machines*.

Sproewitz, A., Tuleu, A., Vespignani, M., Ajallooeian, M., Badri, E., and Ijspeert, A. J. (2013). Towards dynamic trot gait locomotion: Design, control, and experiments with cheetah-cub, a compliant quadruped robot. *The International Journal of Robotics Research*.

Steil, J. J. (2004). Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of the International Joint Conference on Neural Networks*, volume 1, pages 843–848.

Stein, P. S., Stuart, D. G., Grillner, S., and Selverston, A. I. (1999). *Neuron, networks, and motor behavior*. The MIT Press.

Strauss, T., Wustlich, W., and Labahn, R. (2012). Design strategies for weight matrices of echo state networks. *Neural Computation*, 24(12):3246–3276.

Su, H. and McAvoy, T. (1997). Artificial neural network for nonlinear process identification and control. In *Nonlinear process control*, pages 371–428. Prentice-Hall, Inc.

Su, Z. and Khorasani, K. (2001). A neural-network-based controller for a single-link flexible manipulator using the inverse dynamics approach. *IEEE Transactions on Industrial Electronics*, 48(6):1074–1086.

Sun, X.-c., Cui, H.-y., Liu, R.-p., Chen, J.-y., and Liu, Y.-j. (2012). Modeling deterministic echo state network with loop reservoir. *Journal of Zhejiang University SCIENCE C*, 13(9):689–701.

Sussillo, D. and Abbott, L. (2009). Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557.

Sussillo, D. and Abbott, L. (2012). Transferring learning from external to internal weights in echo-state networks with sparse connectivity. *PloS one*, 7(5):e37372.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press.

Suykens, J., De Moor, B., and Vandewalle, J. (2000). Robust local stability of multilayer recurrent neural networks. *IEEE Transactions Neural Networks*, 11(1):222–229.

Suykens, J., Vandewalle, J., and De Moor, B. (1997). Nlq theory: checking and imposing stability of recurrent neural networks for nonlinear modeling. *IEEE Transactions on Signal Processing*, 45(11):2682–2691.

Suykens, J. A. and Osipov, G. V. (2008). Introduction to focus issue: synchronization in complex networks. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3):037101–037101.

Tedrake, R. (2009). Lqr-trees: Feedback motion planning on sparse randomized trees.

Tikhonov, A. and Arsenin, V. Y. (1979). *Methods for solving ill-posed problems*, volume 15. Nauka, Moscow.

Todorov, E., Hu, C., Simpkins, A., and Movellan, J. (2010). Identification and control of a pneumatic robot. In *2010 3rd IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, pages 373–380. IEEE.

Tondu, B. and Lopez, P. (2000). Modeling and control of mckibben artificial muscle robot actuators. *Control Systems, IEEE*, 20(2):15–38.

Tonietti, G., Schiavi, R., and Bicchi, A. (2005). Design and control of a variable stiffness actuator for safe and fast physical human-robot interaction. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005.*, pages 526–531. IEEE.

Torrico, B., Roca, L., Normey-Rico, J., Guzman, J., and Yebra, L. (2010). Robust Nonlinear Predictive Control Applied to a Solar Collector Field in a Solar Desalination Plant. *IEEE Transactions on Control System Technology*, (99):1–10.

Townsend, W. T. and Salisbury, J. K. (1989). Mechanical bandwidth as a guideline to high-performance manipulator design. In *1989 IEEE International Conference on Robotics and Automation, 1989. Proceedings.*, pages 1390–1395. IEEE.

Townsend, W. T. and Salisbury, J. K. (1993). Mechanical design for whole-arm manipulation. In *Robots and Biological Systems: Towards a New Bionics?*, pages 153–164. Springer.

Trebatickỳ, P. (2009). Prediction of dynamical systems by recurrent neural networks.

Tresch, M. and Bizzi, E. (1999). Responses to spinal microstimulation in the chronically spinalized rat and their relationship to spinal systems activated by low threshold cutaneous stimulation. *Experimental brain research*, 129(3):401–416.

Triefenbach, F., Jalalvand, A., Demuynck, K., and Martens, J.-P. (2013). Acoustic modeling with hierarchical reservoirs. *IEEE Transactions on Audio, Speech and Language Processing*.

Triefenbach, F., Jalalvand, A., Schrauwen, B., and Martens, J.-P. (2010). Phoneme recognition with large hierarchical reservoirs. *Advances in neural information processing systems*, 23:2307–2315.

Tsagarakis, N. G., Laffranchi, M., Vanderborght, B., and Caldwell, D. G. (2009). A compact soft actuator unit for small scale human friendly robots. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 4356–4362. IEEE.

Tsagarakis, N. G., Morfey, S., Medrano Cerda, G., Zhibin, L., and Caldwell, D. G. (2013). Compliant humanoid coman: Optimal joint stiffness tuning for modal frequency control. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 673–678. IEEE.

Tyukin, I., Prokhorov, D., and Terekhov, V. (2003). Adaptive control with nonconvex parameterization. *IEEE Transactions on Automatic Control*, 48(4):554–567.

Van Damme, M., Beyl, P., Vanderborght, B., Versluys, R., Van Ham, R., Vanderniepen, I., Daerden, F., and Lefeber, D. (2010). The safety of a robot actuated by pneumatic muscle: A case study. *International Journal of Social Robotics*, 2(3):289–303.

Van Damme, M., Vanderborght, B., Verrelst, B., Van Ham, R., Daerden, F., and Lefeber, D. (2009). Proxy-based sliding mode control of a planar pneumatic manipulator. *The International Journal of Robotics Research*, 28(2):266–284.

Van Der Linde, R. Q. (1998). Active leg compliance for passive walking. In *IEEE International Conference on Robotics and Automation, 1998. ICRA'98.*, volume 3, pages 2339–2344. IEEE.

van der Smagt, P. (1995). Visual robot arm guidance using neural networks.

Van Ham, R., Vanderborght, B., Van Damme, M., Verrelst, B., and Lefeber, D. (2007). Maccepa, the mechanically adjustable compliance and controllable equilibrium position actuator: Design and implementation in a biped robot. *Robotics and Autonomous Systems*, 55(10):761–768.

Vanderborght, B. (2010). *Dynamic stabilisation of the biped Lucy powered by actuators with controllable stiffness*, volume 63. Springer.

Vanderborght, B., Albu-Schaeffer, A., Bicchi, A., Burdet, E., Caldwell, D., Carloni, R., Catalano, M., Eiberger, O., Friedl, W., Ganesh, G., et al. (2013). Variable impedance actuators: a review. *Robotics and Autonomous Systems*, 61(12):1601–1614.

Vanderborght, B., Tsagarakis, N. G., Semini, C., Van Ham, R., and Caldwell, D. G. (2009a). Maccepa 2.0: Adjustable compliant actuator with stiffening characteristic for energy efficient hopping. In *IEEE International Conference on Robotics and Automation, 2009. ICRA'09.*, pages 544–549. IEEE.

Vanderborght, B., Van Ham, R., Lefeber, D., Sugar, T. G., and Hollander, K. W. (2009b). Comparison of mechanical design and energy consumption of adaptable, passive-compliant actuators. *The International Journal of Robotics Research*, 28(1):90–103.

Vandoorne, K., Dierckx, W., Schrauwen, B., Verstraeten, D., Baets, R., Bienstman, P., and Van Campenhout, J. (2008). Toward optical signal processing using photonic reservoir computing. *Optics Express*, 16(15):11182–11192.

Versace, J. (1971). A review of the severity index. pages 771–796.

Verstraeten, D., Schrauwen, B., D'Haene, M., and Stroobandt, D. (2007). An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403.

Verstraeten, D., Schrauwen, B., and Stroobandt, D. (2005). Reservoir computing with stochastic bitstream neurons. In *Proceedings of the 16th annual Prorisc workshop*, pages 454–459.

Verstraeten, D., Schrauwen, B., and Stroobandt, D. (2006). Reservoir-based techniques for speech recognition. In *Proceedings of the World Conference on Computational Intelligence*, pages 1050–1053.

Vischer, D. and Khatib, O. (1995). Design and development of high-performance torque-controlled joints. *IEEE Transactions on Robotics and Automation*, 11(4):537–544.

Wang, C. and Hill, D. (2006). Learning from neural control. *IEEE Transactions Neural Networks*, 17(1):130–146.

Wang, C. and Hill, D. (2007). Deterministic learning and rapid dynamical pattern recognition. *IEEE Transactions Neural Networks*, 18(3):617–630.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Wolpert, D. and Kawato, M. (1998). Multiple paired forward and inverse models for motor control. *Neural Networks*, 11(7):1317–1329.

wyffels, F. (2013). *Sequence generation with reservoir computing systems*. PhD thesis, Ghent University.

wyffels, F., Li, J., Waegeman, T., and Schrauwen, B. (2013). Frequency modulation of large oscillatory neural networks. *Biological Cybernetics*. (submitted).

wyffels, F. and Schrauwen, B. (2009). Design of a central pattern generator using reservoir computing for learning human motion. In *Advanced Technologies for Enhanced Quality of Life, 2009. AT-EQUAL'09.*, pages 118–122. IEEE.

wyffels, F. and Schrauwen, B. (2010). A comparative study of reservoir computing strategies for monthly time series prediction. *Neurocomputing*, 73:1958–1964.

wyffels, F., Schrauwen, B., and Stroobandt, D. (2008). Stable output feedback in reservoir computing using ridge regression. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 808–817.

Yang, C., Ge, S., Xiang, C., Chai, T., and Lee, T. (2008). Output feedback NN control for two classes of discrete-time systems with unknown control directions in a unified approach. *IEEE Transactions on Neural Networks*, 19(11):1873–1886.

Yazdizadeh, A., Khorasani, K., and Patel, R. V. (2000). Identification of a two-link flexible manipulator using adaptive time delay neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(1):165–172.

Yershova, A. and LaValle, S. M. (2007). Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157.

Yildiz, I. B., Jaeger, H., and Kiebel, S. J. (2012). Re-visiting the echo state property. *Neural Networks*.

Yoshida, K. (1999). Swing-up control of an inverted pendulum by energy-based methods. In *Proceedings of the 1999 American Control Conference, 1999.*, volume 6, pages 4045–4047. IEEE.

Zinn, M., Khatib, O., Roth, B., and Salisbury, J. K. (2004). Playing it safe [human-friendly robots]. *Robotics and Automation Magazine, IEEE*, 11(2):12–21.

Zunt, D. (2002). Who did actually invent the word" robot" and what does it mean?". *The Karel Čapek website*.