Reservoir Computing Architectures
for Modeling Robot Navigation Systems

Reservoir computing
voor het modelleren van robotnavigatiesystemen

Eric Aislan Antonelo

Promotoren: prof. dr. ir. B. Schrauwen, prof. dr. ir. D. Stroobandt
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 2011 - 2012

UNIVERSITEIT
GENT

ISBN 978-90-8578-456-2
NUR 984
Wettelijk depot: D/2011/10.500/60

Acknowledgments

A number of people have directly or indirectly contributed to the development of this research. The order in which they are mentioned is not necessarily an order of importance. Furthermore, the list of people given below is not exhaustive.

First, I would like to thank prof. Benjamin Schrauwen, a source of insightful explanations and creative ideas, for his close supervision specially at the beginning of this doctorate, and for his critical reviews which were essential for the refinement of this work. Thanks to prof. Dirk Stroobandt for helping with administrative tasks, reviewing my thesis and enabling financial support mainly during the first year of this research. Thanks to prof. Jan Van Campenhout for, although few, but fruitful and discerning discussions on one of my first journal publications as well as on my thesis.

Thanks to David Verstraeten, for his long-lasting will to help me in diverse matters, research-wise with very clarifying explanations and life-in-Ghent-wise specially during my first months in Belgium. Thanks to David, Karel Bruneel and Antonio Rebordao for the important help with moving to new apartments, furniture shopping, etc. Thanks to all colleagues from the Reservoir Lab that made this environment a nice working place and that, in some or another way, helped me during this period: Benjamin, David, Francis, and Michiel D. (at the early times of the Reservoir lab), and Antonio, Fionntan, Joni, Ken, Michiel H., Philemon, Pieter, Pieter-Jan, Sander and Tim later on. Thanks also to the colleagues from the HES lab for the nice time together: Brahim, Craig, Fatma, Karel, Poona, Tom, Wim.

Thanks to Marnix Vermassen, Annelies Hofman, Rita Breems, Ronny Blomme, Michiel Ronsse, Jeroen Ongenae for the logistic support on the lab infrastructure, and all other matters for making this research and the lab function properly.

Thanks to Xavier Dutoit, Dries Van Puymbroeck, Karel Braeckman, Stefan Depeweg and Tim Waegeman for the joint cooperation concerning the application of Reservoir Computing to mobile robot navigation problems.

I am very grateful to the PhD examination board, for the time and effort spent on the reading and assessing of my thesis as well as for the suggestions for improvement and for their interest during the private defense.

I am also thankful to prof. Mauricio Figueiredo, who was my tutor and supervisor during my undergraduate studies in Brazil, and from whom I learned a lot. Thanks for your support!

Muito obrigado em especial à minha família, meus pais Milino Antonelo e Marcia Helena Antonelo, por serem bons exemplos, pelas oportunidades a mim oferecidas de estudar em boa escolas quando muitos no país têm muito menos que isso, e pelo apoio nesta ousada temporada na Bélgica! Obrigado também aos meus irmãos sangue B, Johny e Suzane! Obrigado a meus avós Helga e Augusto, e Paschoalino e Danuta, pelos bons momentos em família. Dedico esta tese em memória a minha avó Helga Raimundo e meu avô Paschoalino Antonelo.

Para concluir, um especial agradecimento a minha esposa Juliana Rodrigues Antonelo, companheira de todos os momentos, alegres ou tristes, nestes anos na Bélgica. Muito obrigado por tudo!

Eric Aislan Antonelo
Ghent, 13 October 2011

This work was supported by the Special Research Fund (BOF) of Ghent University.

Dit werk werd financieel ondersteund door het Bijzonder Onderzoeksfonds van de Universiteit Gent.

PhD Jury

Prof. Rik Van de Walle, voorzitter
Academisch secretaris, Faculteit Ingenieurswetenschappen
Universiteit Gent

Prof. Jan Van Campenhout, secretaris
Vakgroep ELIS, Faculteit Ingenieurswetenschappen
Unversiteit Gent

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

Prof. Dirk Stroobandt, co-promotor
Vakgroep ELIS, Faculteit Ingenieurswetenschappen
Universiteit Gent

Dr. ir. Bart Wyns
Vakgroep EESA, Faculteit Ingenieurswetenschappen
Universiteit Gent

Prof. Bram Vanderborght
Faculteit Ingenieurswetenschappen
Vrije Universiteit Brussel

Prof. Tony Belpaeme
School of Computing, Communications and Electronics
Plymouth University, UK

Prof. Tom Ziemke,
School of Humanities & Informatics
University of Skövde, Sweden

First (private) defense: 6 October 2011, 14:00
Public defense: 28 October 2011, 17:00

Abstract

Robot Navigation Systems Autonomous mobile robots must be able to safely and purposefully navigate in complex dynamic environments, preferentially considering a restricted amount of computational power as well as limited energy consumption. In order to turn these robots into commercially viable domestic products with intelligent, abstract computational capabilities, it is also necessary to use inexpensive sensory apparatus such as a few infra-red distance sensors of limited accuracy. Current state-of-the-art methods for robot localization and navigation require fully equipped robotic platforms usually possessing expensive laser scanners for environment mapping, a considerable amount of computational power, and extensive explicit modeling of the environment and of the task.

This thesis The research presented in this thesis is a step towards creating intelligent autonomous mobile robots with abstract reasoning capabilities using a limited number of very simple raw noisy sensory signals, such as distance sensors. The basic assumption is that the low-dimensional sensory signal can be projected into a high-dimensional dynamic space where learning and computation is performed by linear methods (such as linear regression), overcoming sensor aliasing problems commonly found in robot navigation tasks. This form of computation is known in the literature as Reservoir Computing (RC), and the *Echo State Network* is a particular RC model used in this work and characterized by having the high-dimensional

space implemented by a discrete analog recurrent neural network with fading memory properties. This thesis proposes a number of Reservoir Computing architectures which can be used in a variety of autonomous navigation tasks, by modeling implicit abstract representations of an environment as well as navigation behaviors which can be sequentially executed in the physical environment or simulated as a plan in deliberative goal-directed tasks.

Navigation attractors A navigation attractor is a reactive robot behavior defined by a temporal pattern of sensory-motor coupling through the environment space. Under this scheme, a robot tends to follow a trajectory with attractor-like characteristics in space. These navigation attractors are characterized by being robust to noise and unpredictable events and by having inherent collision avoidance skills. In this work, it is shown that an RC network can model not only one behavior, but multiple navigation behaviors by shifting the operating point of the dynamical reservoir system into different *sub-space attractors* using additional external inputs representing the selected behavior. The sub-space attractors emerge from the coupling existing between the RC network, which controls the autonomous robot, and the environment. All this is achieved under an imitation learning framework which trains the RC network using examples of navigation behaviors generated by a supervisor controller or a human.

Implicit spatial representations From the stream of sensory input given by distance sensors, it is possible to construct implicit spatial representations of an environment by using Reservoir Computing networks. These networks are trained in a supervised way to predict locations at different levels of abstraction, from continuous-valued robot's pose in the global coordinate's frame, to more abstract locations such as small delimited areas and rooms of a robot environment. The high-dimensional reservoir projects the sensory input into a dynamic system space, whose characteristic fading memory disambiguates the sensory space, solving the sensor aliasing problems where multiple different locations generate similar sen-

sory readings from the robot's perspective.

Hierarchical networks for goal-directed navigation It is possible to model navigation attractors and implicit spatial representations with the same type of RC network. By constructing an hierarchical RC architecture which combines the aforementioned modeling skills in two different reservoir modules operating at different timescales, it is possible to achieve complex context-dependent sensory-motor coupling in unknown environments. The general idea is that the network trained to predict the location and orientation of the robot in this architecture can be used to select appropriate navigation attractors according to the current context, by shifting the operating point of the navigation reservoir to a sub-space attractor. As the robot navigates from one room to the next, a corresponding context switch selects a new reactive navigation behavior. This continuous sequence of context switches and reactive behaviors, when combined with an external input indicating the destination room, leads ultimately to a goal-directed navigation system, purely trained in a supervised way with examples of sensory-motor coupling.

Generative modeling of environment-robot dynamics RC networks trained to predict the position of the robot from the sensory signals learns *forward models* of the robot. By using a generative RC network which predicts not only locations but also sensory nodes, it is possible to use the network in the opposite direction for predicting local environmental sensory perceptions from the robot position as input, thus learning an *inverse model*. The implicit map learned by forward models can be made *explicit*, by running the RC network in reverse: predict the local sensory signals given the location of the robot as input (*inverse model*). Moreover, by cutting interference from the environment and letting this generative network run in closed loop by using only their self-predictions which are fed back to the reservoir, it is possible to internally predict future scenarios and behaviors without actually experiencing them in the current environment (a process analogous to *dreaming*), constituting a planning-like capability which

opens new possibilities for deliberative navigation systems.

Unsupervised learning of spatial representations In order to achieve a higher degree of autonomy in the learning process of RC-based navigation systems which use implicit learned models of the environment for goal-directed navigation, a new architecture is proposed. Instead of using linear regression, an unsupervised learning method which extracts slowly-varying output signals from the reservoir states, called *Slow Feature Analysis*, is used to generate self-organized spatial representations at the output layer, without the requirement of labeling training data with the desired locations. It is shown experimentally that the proposed *RC-SFA architecture* is empowered with an unique combination of short-term memory and non-linear transformations which overcomes the *hidden state problem* present in robot navigation tasks. In addition, experiments with simulated and real robots indicate that spatial activations generated by the trained network show similarities to the activations of CA1 hippocampal cells of rats (a specific group of neurons in the hippocampus).

Samenvatting

Robotnavigatiesystemen Autonome mobiele robots moeten veilig en doelgericht kunnen navigeren in complexe dynamische omgevingen, bij voorkeur met inachtname van een beperkte hoeveelheid rekenkracht en een beperkt energieverbruik. Om van deze robots commercieel haalbare huis-en-tuin-producten te maken met de intelligentie en de capaciteiten om abstracte berekeningen te maken, zijn ook goedkope sensoren nodig, bijvoorbeeld zijn infrarood-afstandssensoren met beperkte nauwkeurigheid. De huidig beste methoden voor robotlocalisatie en -navigatie vereisen echter volledig uitgeruste robotplatformen met dure laserscanners om de omgeving in kaart te brengen, een belangrijke hoeveelheid rekenkracht en uitgebreide expliciete modellering van zowel de omgeving als de uit te voeren taak.

Deze thesis Het onderzoek voorgesteld in deze doctoraatsthesis is een stap richting de ontwikkeling van intelligente autonome mobiele robots met abstracte redeneermogelijkheden maar met gebruik van een beperkt aantal zeer eenvoudige sensoren die ruwe en ruizige signalen leveren, zoals bijvoorbeeld in afstandssensoren het geval is. The basisaanname is dat de laagdimensionale sensorsignalen geprojecteerd kunnen worden op een hoogdimensionale dynamische ruimte waarbij leren en berekenen gebeurt aan de hand van lineaire methoden (zoals lineaire regressie). Dit vermijdt problemen van sensorverwarring zoals vaak voorkomt in robotnavigatietaken. Deze manier

van rekenen is in de literatuur gekend als *Reservoir Computing* (RC). Het *Echo State Network* is zo een RC-model dat in dit doctoraatswerk gebruikt wordt. Het wordt gekarakteriseerd door een hoogdimensionale ruimte geïmplementeerd door een discreet analoog teruggekoppeld neuraal netwerk met een uitdovend geheugen. Deze thesis stelt een aantal Reservoir-Computing-architecturen voor die gebruikt kunnen worden in autonome navigatietaken, door impliciet abstracte representaties van een omgeving te modelleren en door navigatiegedrag te leren dat sequentieel uitgevoerd kan worden in de fysieke omgeving, ofwel gesimuleerd als een plan in doelgerichte taken.

Navigatie-attractors Een navigatie-attractor is een reactief robotgedrag, gedefinieerd door een temporeel patroon van sensor-motor-koppeling in de omgevingsruimte. In dit patroon heeft de robot de neiging om een traject te volgen met attractor-achtige karakteristieken. Deze navigatie-attractoren zijn gekarakteriseerd door hun robuustheid voor ruis en onvoorspelbare gebeurtenissen en door hun inherente botsingsvermijdingsvaardigheden. In dit werk tonen we aan dat een RC-netwerk niet enkel één gedrag kan modelleren maar ook verschillende navigatiegedragingen. Dit kan door het werkingspunt van het dynamische reservoirsysteem te verschuiven naar verschillende *subruimte-attractors* door gebruik te maken van bijkomende externe ingangen die het geselecteerde gedrag voorstellen. De subruimte-attractors ontstaan door de koppeling tussen het RC-netwerk, dat de autonome robot controleert, en de omgeving. Dit alles bereiken we in een raamwerk voor imitatieleren dat het RC-netwerk traint door voorbeelden van navigatiegedrag gegenereerd door een opzichter-controller of een mens.

Impliciete ruimtelijke representaties Uit de stroom van sensorgegevens afkomstig van de afstandssensors kunnen we impliciete ruimtelijke representaties van een omgeving samenstellen door het gebruik van Reservoir-Computing-netwerken. Deze netwerken zijn getraind op een gesuperviseerde wijze om locaties op verschillende abstractieniveaus te voorspellen, van de robotpositie met continue waarden in de globale coordinatenruimte tot de meer abstracte locaties zoals kleine afge-

bakende gebieden en omgevingskamers. Het hoogdimensionele reservoir projecteert de sensorgegevens in een dynamischesysteemruimte, waarvan het karakteristieke uitdovend geheugen de sensorruimte expandeert en zo het sensorverwarringsprobleem oplost. Dat probleem bestaat erin dat verschillende locaties voor de robot als gelijkaardig waargenomen sensorgegevens genereren. In een tweede aanpak gebruiken we niet de lineaire regressie maar wel een ongesuperviseerde leermethode die traagvariërende uitgangssignalen uit de reservoirtoestanden extraheert. Deze methode wordt *Slow Feature Analysis* genoemd en wordt gebruikt om autonoom ruimtelijke representaties te vormen op de uitgangslaag, zonder de vereiste labels van trainingsgegevens met de gewenste locaties.

Hierarchische netwerken voor doelgerichte navigatie Het is mogelijk om, met hetzelfde type van RC-netwerk, zowel navigatie-attractors als impliciete ruimtelijke representaties te modelleren. Door een hierarchische RC-architectuur te construeren die de hoger vermelde modelleringsmogelijkheden in twee verschillende reservoirmodules combineert, kunnen we complexe contextafhankelijke sensor-motor-koppeling in ongekende omgevingen bereiken. Het algemeen principe is dat het netwerk, dat getraind werd om de locaties en orientatie van de robot in deze architectuur te voorspellen, gebruikt kan worden om geschikte navigatie-attractors te selecteren volgens de huidige context. Dit gebeurt door het verschuiven van het werkingspunt van het navigatie-reservoir naar een subruimteattractor. Terwijl de robot van de ene naar de volgende kamer navigeert, selecteert een overeenstemmende contextomschakeling een nieuwe reactief navigatiegedrag. Deze voortdurende opeenvolging van contextomschakelingen en reactieve gedragingen, gecombineerd met een externe ingang die de bestemmingskamer aanduidt, leidt uiteindelijk tot een doelgericht navigatiesysteem dat uitsluitend getraind is op basis van imitatieleren.

Generatieve modellering van omgeving-robot-dynamica Door gebruik te maken van een generatief RC-netwerk dat niet enkel locaties voorspelt maar ook sensorsignalen is het

mogelijk het netwerk in de tegenovergestelde richting te gebruiken om locale omgevingspercepties te voorspellen uit de robotpositie. Dit leidt tot een expliciete kaartgeneratie bij gegeven robotpositie. Het interne simulatiemechanisme dat dit generatieve netwerk toelaat autonoom voorspellingen te genereren die teruggekoppeld worden naar het reservoir, laat toe om intern toekomstige scenarios en gedragingen te voorspellen, zonder ze effectief in de huidige omgeving waar te nemen. Dit vormt de mogelijkheid tot planning en opent nieuwe mogelijkheden voor onderhandelende navigatiesystemen.

Ongesuperviseerd leren van ruimtelijke representaties

We willen een hogere graad van autonomie bereiken in het leerproces van RC-gebaseerde navigatiesystemen die gebruik maken van impliciet aangeleerde omgevingsmodellen voor doelgerichte navigatie. Daartoe wordt een nieuwe architectuur voorgesteld. In de plaats van lineaire regressie gebruiken we nu een ongesuperviseerde methode om automatisch gevormde ruimtelijke representaties aan de uitgangslaag te genereren. Deze methode onttrekt traag-variërende uitgangssignalen uit de reservoirtoestanden en wordt *Slow Feature Analysis* genoemd. Bovendien vereist de generatie van de ruimtelijke representaties geen etikettering van trainingsdata met de gewenste locaties. Er wordt experimenteel aangetoond dat de voorgestelde RC-FSA-architectuur de unieke combinatie in zich heeft van korte-termijngeheugen en niet-lineaire transformaties. Daardoor vermijdt het het verborgen-toestandsprobleem dat typisch is voor robotnavigatietaken. Experimenten met gesimuleerde en echte robots tonen bovendien aan dat ruimtelijke activaties gegenereerd door het getrainde netwerk gelijkenissen vertonen met de activaties van CA1-hippocampale cellen in ratten (een specifieke groep neuronen in de hippocampus).

List of Abbreviations

AI	:	Artificial Intelligence
ANN	:	Artificial Neural Network
BPTT	:	Backpropagation through time
CA1	:	cornu Ammonis 1, a histological division of the hippocampus
EE	:	Environment Exploration behavior
EC	:	Entorhinal Cortex
ESN	:	Echo State Network
ESP	:	Echo State Property
GRC	:	Generative Reservoir Computing (architecture)
HRC	:	Hierarchical Reservoir Computing (controller)
ICA	:	Independent Component Analysis
INASY	:	Intelligent autonomous NAvigation SYstem
LSM	:	Liquid State Machine
LSTM	:	Long Short-Term Memory
MLP	:	Multi-layer perceptrons
NMSE	:	Normalised Mean Square Error
NRMSE	:	Normalised Root Mean Square Error
PC	:	Principal Component
PCA	:	Principal Component Analysis

POMDP	:	Partially Observable Markov Decision Process
RECNA	:	REservoir Computing-based NAvigation system
RC	:	Reservoir Computing
RC-SFA	:	unsupervised learning with Reservoir Computing and Slow Feature Analysis
RL	:	Reinforcement Learning
RNN	:	Recurrent Neural Networks
RTRL	:	Real Time Recurrent Learning
SARSA	:	State-Action-Reward-State-Action
SFA	:	Slow Feature Analysis
SINAR	:	SImulador de Navegação Autônoma de Robôs (Autonomous robot navigation simulator)
SLAM	:	Simultaneous Localization And Mapping
TS	:	Target Seeking behavior

Contents

1 Introduction

This PhD thesis proposes a new efficient and biologically inspired way of modeling navigation tasks for autonomous mobile robots having restrictions on cost, energy consumption, and computational complexity. It is based on the recently proposed Reservoir Computing approach for training Recurrent Neural Networks. In this chapter, first, a brief introduction is given on top-down and bottom-up approaches to Artificial Intelligence as well as to connectionist models. Then, I describe the main types of robot navigation systems, giving an overview on some of such systems. Afterwards, an introductory section on modeling navigation with dynamical systems is presented, which serves as a base for the next chapters in this thesis. To conclude, the main contributions and the structure of this work are presented.

1.1 Artificial Intelligence

Artificial intelligence has been portrayed to society and in science fiction as a technology which enables the creation of intelligent human-like creatures or robots which possess competences and skills at the same level of human intelligence. However, the reality is that, nowadays, AI has not been able to create a completely true intelligent and autonomous system with a general set of human-like skills.

Current AI systems possess very specific competences, the so-called expert systems, which are of a great value for helping decision making for humans, but are very limited skill-wise and in terms of

robustness and adaptation to dynamic environments. They work very well indeed, but for what they were specifically built for. Once environments get more stochastic and unpredictable, these systems will have difficulties in dealing with unforeseen events which were not taken into account during their design.

1.1.1 Top-down vs. Bottom-up Approaches

Symbolic models in traditional AI systems are based on an explicit symbolic representation of the world stored in a knowledge base. Abstraction and specific competences are explicitly modeled with the help of symbols and inference rules which combine symbol strings into new, cognitively meaningful ones. Thus, learning is achieved through manipulation of symbols, disregarding the substrate or the general architecture which makes that possible: the brain. This is termed the **top-down** approach in AI, which is characterized by modeling cognitive capabilities through explicit *symbolic rules*.

Moreover, symbolic models are strongly based on the concept of abstraction. The "intelligent" part of these systems, i.e., the symbolic manipulation, is completely separate from other modules such as perception and movement generation which are delegated to a less relevant class of skills (Brooks, 1991). But the perception and movement generation competences are actually the hard part to be designed. High-level abstract cognitive competences of the modern human being was only achieved after a very long-term evolution of survival skills of which perception and motor competences are the most relevant. So, living systems evolved **bottom-up**, starting with unicellular organisms and going to more complex multicellular systems, invertebrates, and finally to vertebrates which originated about 525 million years ago. Anatomically modern humans originated a mere $200,000$ years ago.

Thus, in order to achieve a truly intelligent autonomous system it is more reasonable to start from the bottom self-regulating mechanisms of intelligence which govern living beings, such as learning at the neuronal level, and build up increasingly more complex competences for self-preservation, social cooperation and so on.

The usual advantages and prominent features of the bottom-up approach are:

- robustness and **adaptation to changes** in a dynamic environment;
- tolerance to malfunction of some processing units, **robust operation**;
- **learning** increasingly more complex competences from the interaction with the world
- **implicit world representations** are constructed with sensory perceptions given by the interaction with the world

These characteristics imply a few things. First, there is the concept of **embodiment** which is closely related to interaction of the intelligent system with the world. *Embodied cognition* states that body, brain and the world are the main constituting parts of an intelligent system (Wilson, 2002), i.e., that the mind is shaped by the body and by the interaction of the latter with the world. This ultimately leads to the notion of *learning* and *emergent behavior*, a direct result of the interplay between body, brain and the environment. Learning is a capability resulting from the inter-celullar chemical interactions in the brain, where massive parallel interconnections of its basic constituting elements, the neurons, define a distributed knowledge representation system which changes over time as to maximize utility or survival. It is from these learning mechanisms at the cellular level that intelligent behavior as well as abstract implicit world representations are made possible. Additionally, it is not reasonable to completely separate living organisms from the environment they live in, because one affects the other, i.e., both systems change each other through feedbacks: sense => process (possibly change implicit representation) => act => environment (change) => sense and so on.

As a result, intelligent autonomous systems are inherently fit to their body and to their environment. This can also be seen from the perspective of evolution which works by evolving both the physical body and the mind (in the form of innate behaviors) which are subject to environmental pressure. In this way, it is the environment which defines how individuals evolve, learn and understand the world.

The learning capacity of intelligent systems also allows them to **adapt** to a dynamic environment which changes over time and generates unpredictable events. As the system is always learning, it can adapt as well to new environments and new situations without the need to redesign the whole system.

A second point is that the bottom-up approach makes it possible to design inherently **fault-tolerant systems** once they are formed by distributed control modules or processing units (Brooks, 1991), such as neurons in connectionist models. The non-centralized nature of these systems is such that the malfunctioning of some units do not affect the operation of the system as a whole.

A third point refers to the capacity to learn **increasingly more complex behaviors** or competences (Steels, 1993). As a result from evolution, an individual is born with a predefined or innate set of behaviors. From this basic set of unconditioned responses to the environment, an intelligent autonomous system is able to learn more complex responses through *conditioning* (Pavlov, 1927; Skinner, 1953) or even by imitating their peers (Heyes, 1994; Rizzolatti and Craighero, 2004). Hierarchies in the brain, which are innate structures formed by evolutionary processes, can also play an important role in learning more abstract concepts as the neural activity goes to deeper layers in the hierarchies.

This work has its roots at the bottom-up approach for designing intelligent systems, for the reasons explained above.

1.1.2 Connectionist Models

Connectionist models are constituted of interconnected networks of simple processing units. The most common form of connectionism is given by an *artificial neural network* (ANN). As the name suggests, it is an artificial and simplified model of networks found in the human brain. As the networks of neurons in the brain, ANNs are formed by decentralized networks of interconnected units which learn by modification of their connection weights.

These models are used in a variety of fields and applications, including: optical character recognition (LeCun et al., 1989; Simard et al., 2003) (achieving state-of-the-art performance), autonomous flying of aircrafts in situations of failure (Anon, 1999), fault detection in industrial processes (Bishop, 1995) and medical diagnosis

and analysis (Ster et al., 1996).

1.1.2.1 Artificial Neural Networks

The brain is a highly complex, nonlinear and parallel information processing system. Its main constituent cells, known as neurons, are organized in such a way that certain tasks (e.g., pattern recognition, perception and motor control) are computed much faster than in the fastest existent digital computer. It is estimated that there are about 10^{11} neurons in the brain. Neurons collect input signals through structures called dendrites. The input signals are processed in the soma and the resulting output signals are transmitted to other neurons through axons and its ramifications (Fig. 1.1). The area between an axon of a neuron and a dendrite of another neuron is called **synapse**. It has an important role in the transmission of signals between cells. A synapse can alter a signal originating from an axon or it can even obstruct the transit of a signal to a dendrite. Furthermore, synapses can be modified during the flow of signals between neurons, making likely that they are associated with memory and learning (Haykin, 1999).

McCulloch and Pitts proposed the first mathematic neuron model in 1943 (McCulloch and Pitts, 1943). In their work, they describe the mathematics of neural networks. The proposed model is based on the neurophysiologic plausibility of a neuron with a soma function and a threshold, where the weights represent the biological neuron synapses. Later, it was shown that a neural network with a sufficient number of neurons and with a suitable learning procedure could approximate any computable function (Haykin, 1999). Fig. 1.2 shows the formal model proposed by McCulloch and Pitts, where $\mathbf{u} = [u_1, u_2, \cdots, u_n]$ is the input vector and $\mathbf{w} = [w_1, w_2, \cdots, w_n]^T$ is the synaptic weights vector related to the neuron memory. The neuron output is calculated by:

$$y = f\left(\sum_{i=1}^{n} u_i w_i\right) \tag{1.1}$$

where: f is a nonlinear threshold function (the output is high when the sum exceeds a certain limit; otherwise, the output is very low).

There are many models of artificial neural networks in the liter-

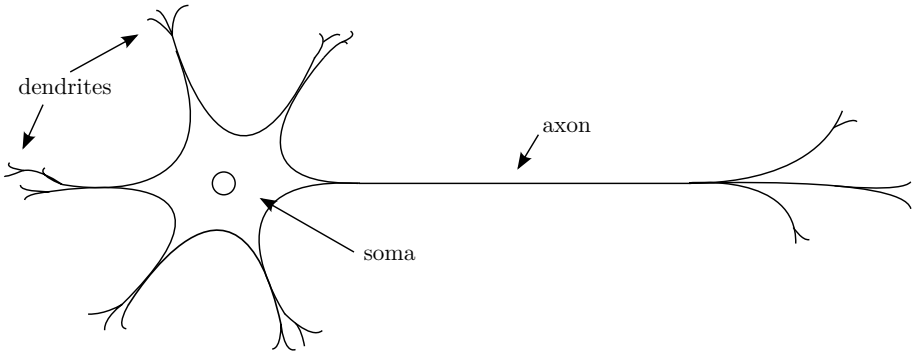


Figure 1.1: Model of biological neuron.

ature. In general, these models have one of the following architectures: one-layer architecture, multi-layer architecture or recurrent architecture (Fig. 1.3). The former two are *feedforward networks*, and are characterized by an unidirectional flow of activity from the input units to the output units. On the other hand, *recurrent networks* have at least one cyclic path connecting the neurons, which means that the previous state of the network influences the current state as in a dynamical system.

Besides the diversity of architectures for neural networks, there is also a variety of *learning algorithms* for changing the weights of the network. One of the methods more disseminated for the learning of multi-layer perceptrons (MLPs) is the gradient-descent method (Rumelhart et al., 1986). The *error backpropagation* method, an efficient way to compute gradients, is executed in the opposite direction to the usual propagation of activity in the network, what makes this method biologically implausible. As this method is based on stochastic gradient-descent, it suffers from slow convergence and the learning process can get stuck in a local error minimum. Furthermore, it is not trivial to choose an appropriate network topology, i.e., the number and size of hidden layers, for a given task. This is because a too simple model, with few hidden units, may cause *underfitting*, while an excessively complex model, with many hidden units, may cause *overfitting*:

Underfitting happens when the prediction performance is poor due to the low number of model parameters with respect to

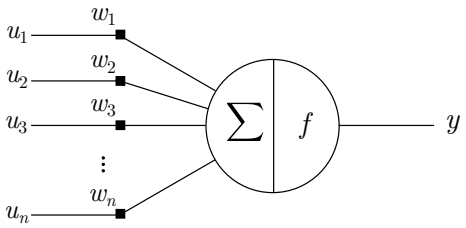


Figure 1.2: Artificial neuron model.

number of training observations.

Overfitting happens when the model describes small random fluctuations in the training set instead of the underlying desired relationship. This causes a poor predictive performance on test (unseen) data and is usually associated with complex models which have too many parameters relative to the number of training observations.

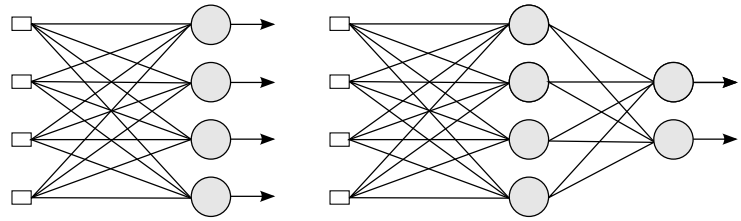
It is also difficult to properly train deep networks (with many layers) with this method, since the gradient vanishes for the deepest layers.

1.1.2.2 Recurrent Neural Networks

While feedforward networks are suitable for implementing static input-output mappings, recurrent neural networks (RNNs) implement dynamical systems and are more biologically plausible.

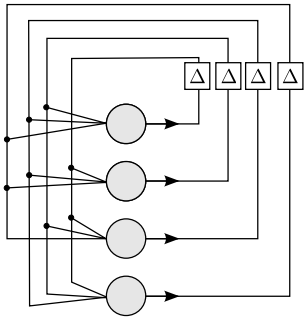
One of the early models of recurrent networks is the *Hopfield network* (Hopfield, 1982), although it is not a general RNN as it is not designed to process sequences of patterns, but it requires stationary inputs instead. Its weight connection matrix is symmetric (to guarantee asymptotic convergence of the energy minimization function), and all units connect to each other, except to themselves (Fig. 1.3(c)). The Hopfield network is usually used as an associative memory, where the patterns to be learned are encoded as stable fixed-point attractors in the state space. Training this network consists of finding the weights which minimize an energy-based cost function (using *Hebbian learning* (Hebb, 1949)).

Some other early simple models of recurrent networks are Jordan (Jordan, 1986) and Elman (Elman, 1990) networks. These networks can maintain contextual information in their states by feeding back



(a) One-layer architecture

(b) Multi-layer architecture



(c) Recurrent architecture (Hopfield network)

Figure 1.3: Architectures of neural networks.

the activations of hidden units (for Elman networks) and output units (for Jordan networks) to input units.

The most used learning method for RNNs is **Backpropagation through time** (BPTT) (Werbos, 1990). This method unfolds the network in time, removing recurrences by creating several identical layers representing consecutive timesteps, and then performing standard error back-propagation on this unfolded network for the computation of gradients.

The known problems with BPTT originates partially from the error backpropagation method based on gradient-descent: slow convergence issues, and if the learning converges, it is not guaranteed to find the global optimum. Additionally, bifurcations may occur during the training process, which could even cause an unexpected growth of the error in the vicinity of bifurcations (Doya, 1992). In practice, only small networks of up to 20 units are used, and memory spans are usually limited to about 20 timesteps due to the fact that the backpropagated error gradient vanishes exponentially over time (Bengio Y., 1994).

Another method used to train RNNs is **real time recurrent learning** (RTRL) (Williams and Zipser, 1989; Doya, 2002). It is used in online learning tasks as it computes the error gradient at every timestep. As it is very computationally expensive, its use is limited to very small networks.

1.1.2.3 Reservoir Computing

In order to overcome the downsides of traditional RNN training such as BPTT and RTRL, a novel paradigm of computation with dynamical systems, namely *Reservoir Computing* (RC), has been proposed in (Verstraeten et al., 2007) which can be utilized to achieve efficient training of recurrent neural networks. RC-based systems possess two parts: a recurrent non-linear layer, called *reservoir*, and a linear readout output layer (Fig. 1.4). The main aspect of an RC network is that the recurrent connections of the reservoir are fixed, whereas the readout output weights are the only ones that are trained. This characteristic simplifies a lot the training of recurrent networks, as any standard classification or regression method can be used to train the output layer. In practice, typically linear methods are chosen as they ensure convergence of the training process to a global optimum.

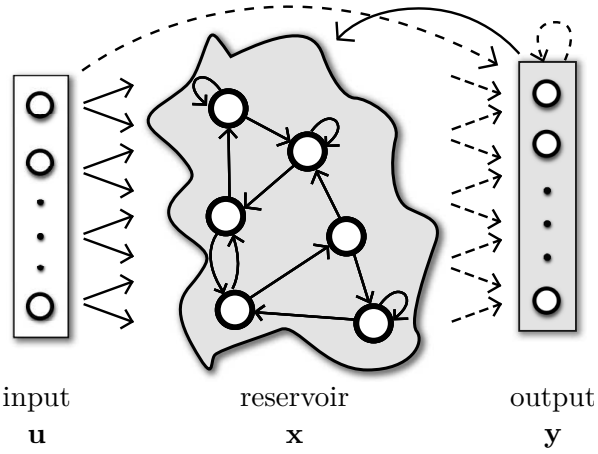


Figure 1.4: Reservoir Computing (RC) network. The reservoir is a non-linear dynamical system usually composed of recurrent sigmoid units. It projects the input into a high-dimensional non-linear space, where it is easier to apply linear regression or classification. Temporal, context-dependent tasks such as time series prediction are well handled by RC. Solid lines represent fixed, randomly generated connections, while dashed lines represent trainable or adaptive weights.

Two RC models for RNN training have been invented independently: the *Echo State Network* (ESN) (Jaeger, 2001) and the *Liquid State Machine* (LSM) (Maass et al., 2002). While the former applies for networks of analog sigmoid neurons as reservoirs, the latter is more general but typically uses spiking neural networks in the reservoir layer. An even earlier RC model applied to sensory-motor sequence learning has been proposed in (Dominey, 1995).

1.2 Robot Navigation Systems

The exploration of unknown environments in the real world has constantly caused a great appeal not only to the scientific community, but also to the humanity in general. The implementation of mobile robots for such use is not an easy task.

Most of current mobile robots operate in simple and controlled environments. In general, these robots are assisted by placement of special landmarks in the target environment, indicating the path to be followed. In this context, the development of autonomous robots, which are adaptive to complex and unknown environments,

is a reason of intense research.

Usually it is possible to understand mobile robot navigation as a problem of establishing trajectories, so that tasks (goals) are accomplished with acceptable performance. The vehicle tasks could be, for example: to capture targets, to deviate from obstacles, to follow walls, to recharge batteries, to exploit the environment, etc.

Important applications come up from the development of mobile robots that are capable of operating in complex environments: surface cleaning (floors, industrial tanks, ship hulls, external structures of buildings, ducts, etc.), transport of materials, e.g., crop of agricultural products, and vigilance systems.

Research about navigation systems has been done in several ways, depending on the characteristics of the environment, the robot model, the type of the task, and the performance criteria (Arkin, 1998). A class of navigation systems, the autonomous systems, has captivated the scientific community not only because of the challenge involved but also because of the strategic importance. Such systems generate the robot trajectory in an unknown environment without external help.

1.2.1 Early models of deliberative systems

Standard models of deliberative systems for autonomous navigation rely on a predefined set of rules for path planning, under the *sense-model-plan-act* framework. A lot of design effort has to be put in creating a map of the environment and modeling all possible events and situations during robot navigation. These systems also usually present a planning program which generates the complete trajectory a priori, which can be computationally expensive.

The first generation of mobile robots endowed with a navigation system comes from the sixties. Shakey, a mobile robot developed by Nilsson in 1969 (Nilsson, 1984), had a navigation system that worked in an environment with special demarcated objects. Its perception module provided information for another module that used symbolic inference rules to generate navigation decisions.

Other robots were also based on symbolic processing in the first decades of research on the area: Hilare in 1977 at Laboratoire d'Automatique et d'Analyse des Systèmes, France (Giralt et al., 1984), Cart in Stanford (Moravec, 1977), and Rover at CMU

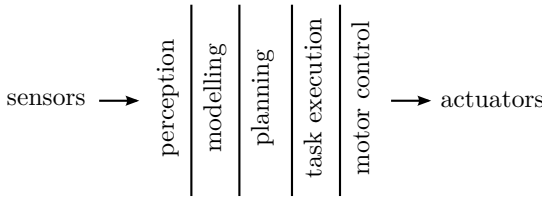


Figure 1.5: Sequential (functional) modules execution for a mobile robot control system.

(Moravec, 1983). Such robots based their decisions on an internal world representation that was used for path planning, characterizing them as deliberative systems (Fig. 1.5).

1.2.2 SLAM

SLAM stands for *Simultaneous Localization And Mapping*. One way to achieve purposeful navigation is the ability for self-localization of an autonomous robot in its environment. Markov localization (Thrun et al., 2005) or Kalman filtering techniques (Siegwart and Nourbakhsh, 2004)) for solving the SLAM problem represents the state-of-the-art in simultaneous localization and mapping (Bailey and Durrant-Whyte, 2006), but usually requires expensive laser-range scanners[1], and have to take into account the following points: the modeling of the noise of each sensor, the kinematic model of the robot, the costly drawing of a priori map of the environment or a mechanism for map building during navigation, matching sensory data to the stored map representation for the correction of position estimation, and so on.

In traditional localization algorithms, the representation of the stored map can be grid-based, topological, or hybrids (Siegwart and Nourbakhsh, 2004). Grid-based methods produce metric maps and have high resolution, while topological maps are more abstract and describe the environment as a graph of connected nodes. The advantages of probabilistic models are: easy human interpretation of the robot position in the stored map; accurate descriptions of the map for grid-based methods (Siegwart and Nourbakhsh, 2004); and ef-

[1]There are few works in the literature using SLAM for small mobile robots with few infra-red sensors (Caprari et al., 2001); and visual SLAM with minimal sensing (using a camera) and computational requirements (Andreasson et al., 2007).

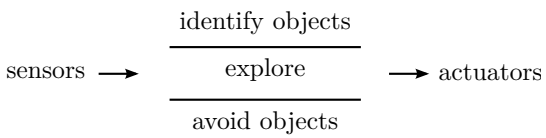


Figure 1.6: Layers of behaviors for a mobile robot control system.

ficient planning in topological maps (Thrun et al., 2005). Given some assumptions, state-of-the-art algorithms, such as the FastSLAM method (Thrun et al., 2005), are able to solve the SLAM problem.

The SLAM approach lacks in adaptation and learning capabilities and usually requires fully equipped robot platforms with expensive laser scanners of high-precision for environment mapping[2]. Furthermore, it *usually* has rather large computational requirements. This sensory apparatus is currently still very expensive, consume a lot of power, and cannot easily be applied in small mobile robots with very limited computational and power requirements.

1.2.3 Behavior-based approach

Behavior-based approaches to robotics have been proposed early in the literature (Brooks, 1986, 1991; Arkin, 1998). Instead of having several modules for perception, world modeling, planning and execution, they are based on individual intelligent control modules, where each one contributes to behavior generation for controlling a robot.

Rodney Brooks suggested an architecture for controlling mobile robots in 1986 (Brooks, 1986), called Subsumption Architecture. This model is based on task-achieving behaviors organized in different layers of control. Each class of behaviors is implemented in a specific layer (Fig. 1.6).

Brooks defines a level of competence as an informal specification of a class of behaviors that the robot can present. Each such level is implemented in a layer. Depending on the current situation, a higher level of competence can suppress the output of lower levels of competence in order to attend the current robot priorities.

[2]However, it is known that laser scanners tend to become cheaper as its technology evolves and becomes more common in everyday life.

The system designed by Brooks aims at robustness (the system should work even when some sensors fail or when the environment changes drastically) and extensibility (more capabilities can be added to a robot). Besides, it is a behavior-based system and, unlike deliberative systems, is not based on cognitive processes on an internal world representation. Interestingly, there is no unique agreement on the difference between reactive and behavior-based systems. Mataric (2001) presents distinct characteristics for both classes of systems. On the other hand, Arkin (1998) considers behavior-based systems as a specific class of reactive systems.

Autonomous robots should be able to learn their abilities through interaction with the environment. Learning its own internal rules for sensory-motor coupling in close interaction with the environment represents a higher degree of autonomy for a robot. This also implies adaptation and robustness to noise and unpredictable events.

1.2.4 Biologically inspired navigation systems

1.2.4.1 Place cells

Experiments accomplished on rats show that their hippocampus forms activation patterns that are associated with locations visited by the rat. These so called *place cells* encode the spatial location of the animal into its environment. They fire whenever the animal is in a particular location (O'Keefe and Dostrovsky, 1971; Moser et al., 2008), which defines the *place field* of the cell.

Another type of spatial activation is found in grid cells from the entorhinal cortex. The activation of these cells have shown to follow a grid pattern in circular and square environments and probably have an important role in the formation of place cells (Moser et al., 2008).

A third type of spatial encoding cells are the head-direction cells, whose activation is dependent on the rat's head direction while being position-invariant.

It is assumed that two classes of input are available for spatial encoding cells: idiothetic and allothetic. While allothetic inputs are originated from the external environment (vision, tactile and auditory signals), idiothetic input could be formed by self-motion (proprioceptive) signals and encoders which indicate how the body's

internal state evolves. The origin of these idiothetic signals and the mechanisms for the integration of these signals with allothetic input have not yet been determined (Moser et al., 2008).

1.2.4.2 Overview on existing navigation systems

There are several works in the literature which employ Recurrent Neural Networks (RNNs) for designing localization and navigation systems for mobile robots. In (Tani, 1996), RNNs are used for model-based learning in robot navigation. In order to achieve situatedness during navigation, a forward model of the mobile robot is learned in a self-organized way using Backpropagation-through time. The internal model predicts the next sensory input given the current sensors (range image and travel distance) and the motor output. In this way, it learns to be situated through interaction with the environment by learning the environmental attractor in the offline training phase. Other early related works for situated robotics are Verschure et al. (1992) and more recently Verschure et al. (2003).

Floreano and Mondada (1996) tackle evolutionary strategies for RNNs in the context of a homing navigation task. In their work, a RNN is evolved so that a mobile robot drives as long as possible around an arena and goes back to a recharging area whenever its battery level is near empty. The evolved RNN learned an internal representation which is a function of the robot position and of the battery level.

Other models of hippocampal place cells and biologically-inspired navigation exist in the literature. In Arleo et al. (2004), unsupervised growing networks are used to build an architecture with idiothetic and allothetic components that are combined in a hippocampal place cell layer to support spatial navigation (validated using a Khepera mobile robot with 2D vision sensors). Their model explicitly uses dead-reckoning to track the robot position and associates place cell firing with the estimated position.

In Milford (2008), a hippocampal place cell model is designed to solve the SLAM problem. They choose a pragmatic approach, favoring functionality over biologically plausibility. Their model, called RatSLAM, has a 3D structure for pose cells (representing beliefs for the robot position and orientation) which learn associative connections with view cells (allothetic representation). They validate

their model with several mobile robots, equipped with a camera, in indoor and outdoor environments. Other works oriented towards modeling an animal's capability for spatial navigation are given in (Stroesslin et al., 2005; Chavarriaga et al., 2005). A single learning technique which maximizes slowness of the output signal applied to hierarchical networks is able to generate self-organized representations of place cells as well as of head-direction cells (Franzius et al., 2007a) without odometry information. A similar method based on temporal stability for learning hippocampal place cells for mobile robots is given in (Wyss et al., 2006).

Most of these models are based on rich visual (pixel-based) stimuli as external sensory input and/or use odometry for path integration. For a further review on biologically-inspired localization models, see Filliat and Meyer (2003) and Trullier et al. (1997).

1.2.5 Towards energy efficient navigation systems

This thesis aims at designing intelligent navigation systems from a bottom-up perspective, where learning of implicit world representations and complex sensory-motor coupling is inspired on the implicit, basic mechanisms of intelligence which control biological systems. Thus, an essential requirement is that these intelligent systems process information and become *situated* in the environment (Wilson, 2002) by solely using its local view of the environment given by the sensory apparatus present in the agent or robot.

For that, this work uses a biologically plausible recurrent neural network based system which can be efficiently trained under the recently emerging paradigm of Reservoir Computing (RC). Training in RC-based systems gets simplified as the recurrent non-linear part (called reservoir) is left fixed, while the readout output weights are the only part to be trained with standard linear regression techniques.

It is advocated in this thesis that RC networks can be used in a diversity of modeling and navigation tasks by considering **small and energy efficient autonomous mobile robots**, an important research topic in the field of robotics. This is mainly due to their near-term real world applicability as domestic service robots, such as e.g. the small and inexpensive iRobot service robots. These robots must be easy to *teach* using training sequences generated by a su-

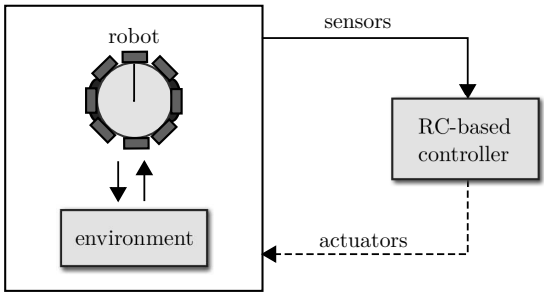


Figure 1.7: Robot controller and environment are coupled systems. A reactive behavior yields a navigation attractor in the environment space. Dashed connections are trainable.

pervisor or even their behaviors could be evolved in a reinforcement learning way. In order to make them function properly in complex and dynamic environments, they need to be aware of their position as well as to create models of the environment for deliberative planning. All of this has to be performed using a **limited number of distance sensors with low accuracy, and with a restricted amount of computational power**.

1.3 Modeling Navigation via Dynamical Systems

This section shows the different possibilities and procedures for modeling autonomous navigation systems with Reservoir Computing (RC) networks.

1.3.1 Navigation attractors

The intelligent navigation systems in this work are designed according to the notion of navigation attractor.[3] By viewing a reactive robot controller and its environment as coupled systems (Fig. 1.7), the robot trajectory resulting from the autonomous navigation forms a sequential **sensory-motor pattern throughout the environment space** (Fig. 1.8). Once projected into the dynamical system space of the reservoir by stimulating it with sensory signals, the

[3]The term *attractor* is used in this thesis more metaphorically and does not directly relate to the exact definition of attractor in mathematics.

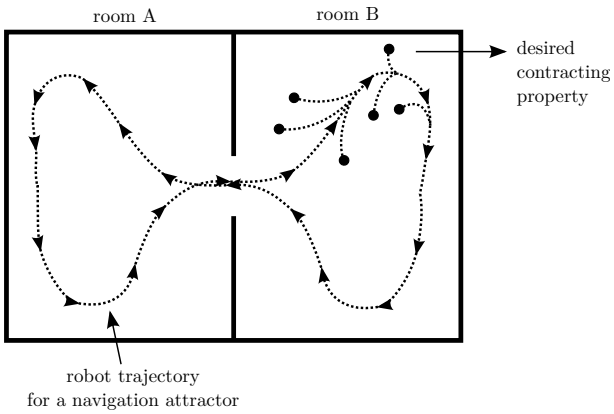


Figure 1.8: Representation of a reactive navigation attractor or behavior in the environment space and desired contracting property.

navigation attractors become *embedded into the network* through the coupling of RC network (controller) and environment.

A desired property for reactive navigation attractors is that the robot converges to an *attractor trajectory* if placed in another location of the environment (Fig. 1.8). Moreover, these reactive behaviors should inherently avoid obstacles so that dynamic obstacles could be circumvented, by moving away from the attractor trajectory and, in the sequence, moving back to the attractor. This confers robustness capabilities for the controller.

The training of an RC network for modeling a navigation attractor can be achieved through supervised learning, i.e., by observing a supervisor controller which provides a desired input-output (sensory-motor) coupling representing the target behavior over time.

1.3.2 Context-based Navigation Attractors

For empowering navigation systems with a more complex and high-level behavior, it is necessary to simultaneously learn multiple reactive navigation attractors. By decomposing a complex behavior into simpler ones, such as in Fig. 1.9, it is possible to create execution plans defining a sequence of task-achieving behaviors as desired (the simpler behaviors are navigation attractors as defined in the previous section).

In order to embed multiple reactive behaviors into a single RC

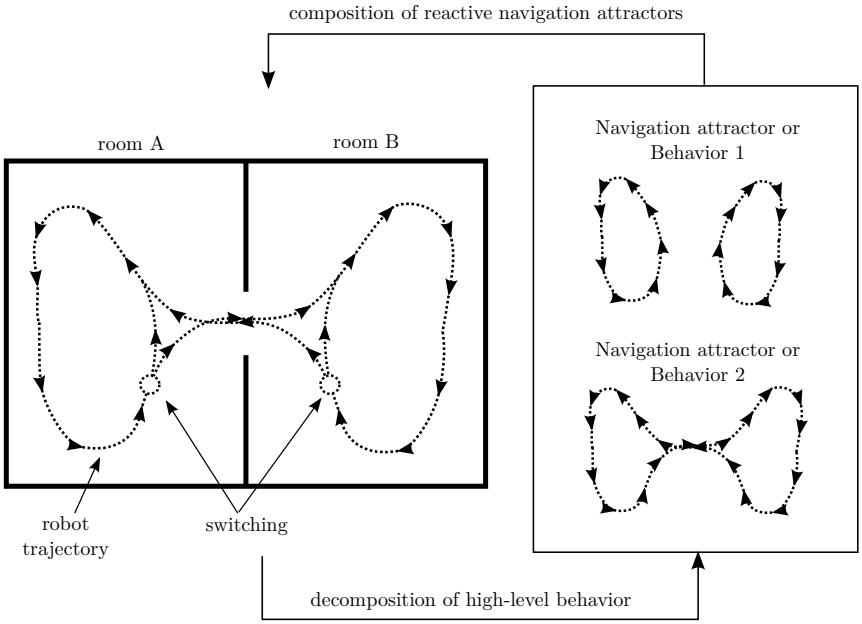


Figure 1.9: Schematic view of decomposition of high-level behavior into reactive navigation attractors.

network, it is necessary to add external binary inputs to the RC network (Fig. 1.10), capable of shifting the attractor dynamics to a confined sub-space corresponding to the selected behavior. The external input acts as a bias during the execution of a reactive behavior. A switch to a different behavior will cause a shift into a different operating point of the reservoir, which in turn is coupled to the environment.

As this architecture (Fig. 1.10) is trained using linear regression on the dynamical system space (only the motor actuators given by the dashed connections are trained), the shift in the high-dimensional space caused by the external binary input (which can be a binary vector for multiple behaviors) makes possible that a linear discrimination by the readout output layer is sufficient to confine navigation attractors to different sub-spaces (Fig. 1.11). Thus, this architecture supports the simultaneous learning of many (even conflicting) behaviors by the trick of shifting the reservoir state space. The number of behaviors that could be learned is limited by the mem-

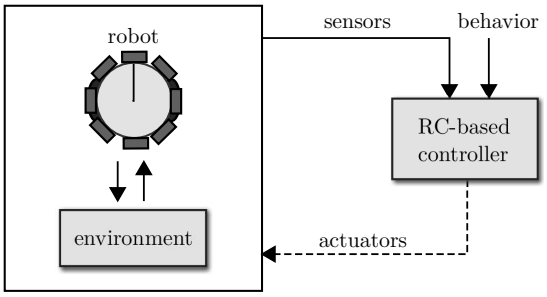


Figure 1.10: Modeling multiple reactive behaviors or navigation attractors using a single RC network via external binary input channel. Dashed connections are trainable.

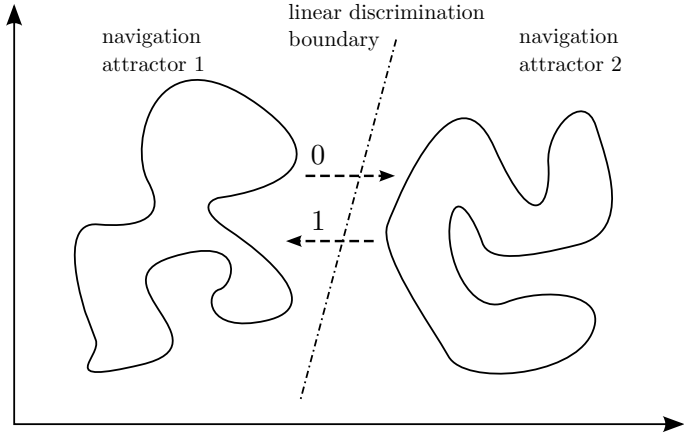


Figure 1.11: Example of two navigation attractors in bi-dimensional dynamical system space. Dashed arrows represent switching events caused by activities of external input channels.

ory capacity[4] of the network, and determines the size of the external input vector.

The autonomous coordination of the switches between navigation attractors may be learned by elaborating an extra hidden layer which is responsible for autonomously detecting the right time to switch (Fig. 1.12). This layer should be trained to generate context switches which indicate a change to a specific navigation attrac-

[4]The memory capacity (MC) is the capacity of an ESN with linear output units to reproduce the past input stimulus from the current state. MC is bounded by the number of neurons n_{res} in the reservoir. A more formal definition is given in (Jaeger, 2002a). More nonlinear reservoirs have less memory capacity than linear reservoirs.

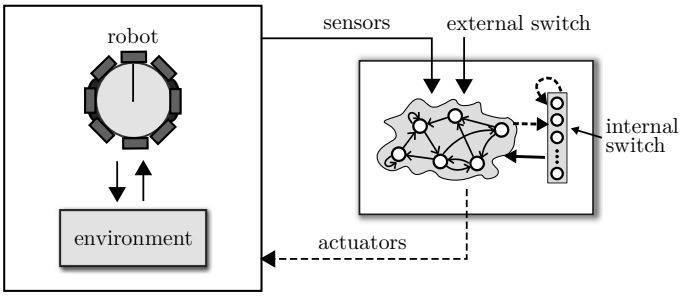


Figure 1.12: Hierarchical networks composed of a reservoir and a trainable hidden layer for modeling complex sensory-motor coupling based on external and internal switches. Dashed connections are trainable.

tor for fulfilling the goals of the high-level navigation task. Now, not only external input, but also activities generated at a hidden layer can provoke context switches between navigation attractors (Fig. 1.13).

If this trainable hidden layer can, for instance, model the room in which the robot is situated (where each unit represents the detection of a specific room, thus a binary value), then, in principle, for each room a specific navigation attractor or behavior can be generated. Under this scheme of context-based navigation, a final destination room would be given as an external input. Each combination of (external and internal) switches sets the dynamical system (reservoir) at a confined space, a **sub-attractor** (i.e., context-based navigation attractor) corresponding to a specific coupling between RC-based controller and environment (Fig. 1.13). By training this architecture in a supervised way with examples of robot trajectories, an autonomous goal-directed navigation system results from the composition of low-level reactive behaviors in a sequence controlled by the context switches.

1.3.3 Disambiguation of incomplete state in the dynamical system space

The importance of training on the dynamical system space is that the history of sensory readings is encoded in the trajectory of the dynamical system. While the memoryless sensory space is susceptible to **sensor aliasing** problems, the dynamical reservoir system

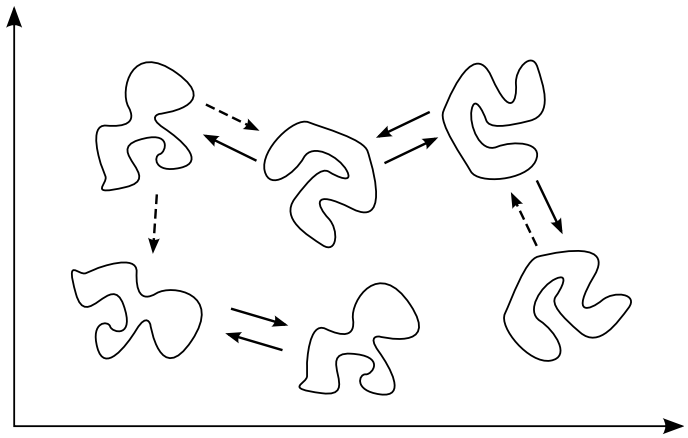


Figure 1.13: Example of multiple navigation attractors in simplified bi-dimensional dynamical system space. Solid arrows represent internal switching events (context switches) (Fig. 1.12) while dashed arrows represent switching events caused by activities of external input channels.

has an intrinsic fading memory which overcomes ambiguities present in the local sensory input space.

This can be viewed more clearly in Fig. 1.14, which shows the sensory readings and the corresponding dynamical system trajectory considering three iterations of a hypothetical robot wandering in an environment. After the $t = 1$, there are two different paths that the robot follows, which actually drive the dynamical system to different points in state space. At $t = 3$, the robot gets similar sensory readings in both situations while being at distinct locations in the environment. Whereas the sensory space is ambiguous because two different positions yield similar perceptions, the dynamical system encodes the robot path in its trajectory, disambiguating the local sensory perception.

An important application of this is **robot localization**, which is made possible by the projection of a low-dimensional input space which provides incomplete information about the environment into a high-dimensional, nonlinear space with temporal processing capabilities where the location can be linearly extracted or discriminated.

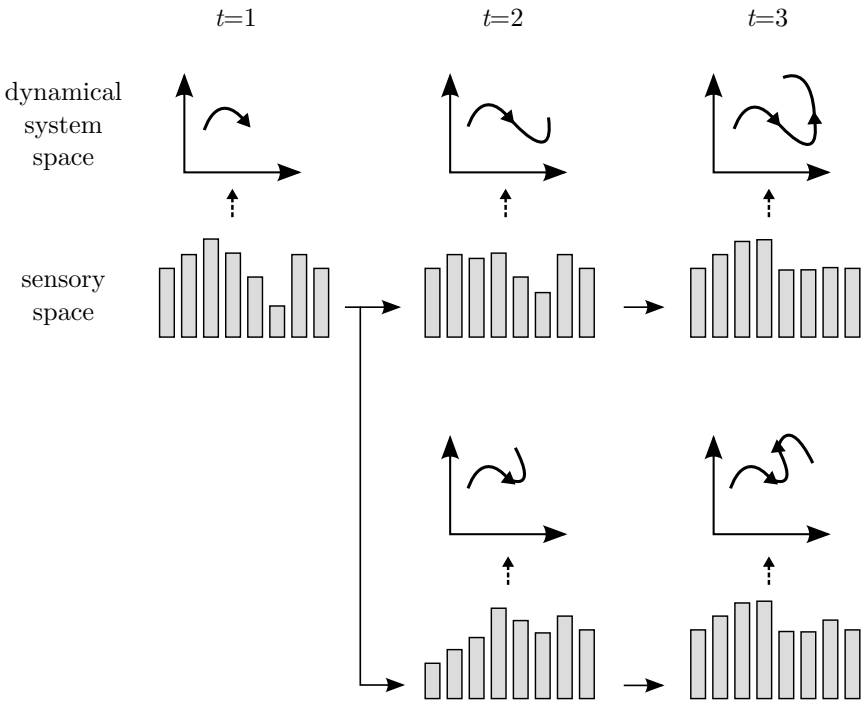


Figure 1.14: Comparison of dynamical system space and sensory space for two different paths chosen by a robot. The dynamical system starts at the same state at $t = 1$. At $t = 3$, the robot is at distinct locations, which is associated to different states in the dynamical system space while the sensory readings are very similar. Whereas the sensory space is memoryless and susceptible to *sensor aliasing* problems, the dynamical system space has an inherent memory which can be used, for instance, to predict the robot location in the environment or to perform context-dependent behaviors.

1.3.4 Timescales and Hierarchies

The dynamical system implemented by the recurrent reservoir is usually set to operate at the edge of stability for exhibiting rich dynamics. The speed of the reservoir dynamics can be controlled by the use of leak rates (Jaeger et al., 2007), which basically enables individual low-pass filters for each neuron in the reservoir, creating a internal memory at the neuronal level.

In practice, a single dynamical reservoir system is not able to model navigation and localization competences simultaneously with acceptable performance. This is due to the multiplicity of timescales present in the sensory signal (e.g., when the robot shows different speeds) and mainly when each task space requires a different timescale.

Navigation attractors are competences which require fast dynamics in order to cope with quickly changing environments and unpredictable events. On the other hand, a dynamical system used to detect the current robot room in a big multi-room environment should operate at a slower timescale when compared to a navigation task. This is because context switches provoked by leaving a room and entering another one do not happen frequently.

In this way, for a goal-directed navigation system based on context switches (Fig. 1.12) to work properly in practice, there must be a hierarchical architecture with two reservoir networks operating at different timescales, one to model the navigation attractor and another one responsible for generating the context switches.

1.3.5 Internal Predictive Simulation of Behaviors

In deliberative navigation systems, it is also necessary to perform path planning for achieving goal-directed navigation. In this work, this is done by predicting the future outcome of reactive behaviors. By coupling the dynamical system with self-predictions of sensory perceptions and of the robot location, for instance, future scenarios can be internally simulated based on the reactive behavior present in the training data. This **generative architecture** is given in Fig. 1.15. All sensory and context nodes are trained.

This internal closed loop simulation generates a predictive sequence of robot locations and sensory perceptions which defines a

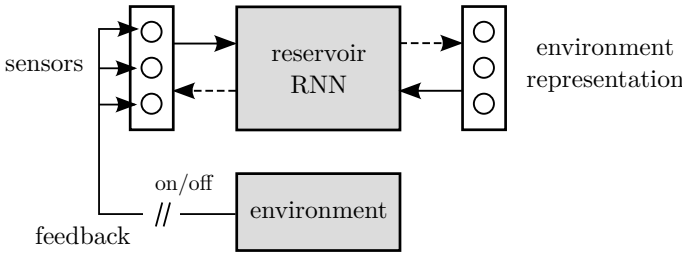


Figure 1.15: Generative architecture for modeling reactive behaviors which could be used for path planning. Both sensory nodes and context nodes (e.g., for localization) are trained in a supervised way by using feedback from the environment. During testing, this feedback can be closed such that only self-predictions are used to stimulate the dynamical system. Only dashed connections are trained.

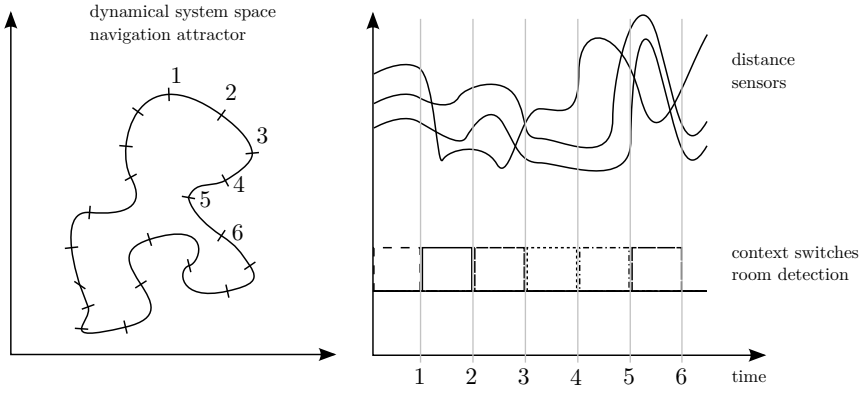


Figure 1.16: Example of navigation attractor in simplified bi-dimensional dynamical system space (left) generated by the internal autonomous simulation of a reactive behavior, executed by feeding back the self-predictions of sensory nodes and context switches (robot location) (right) for the architecture of Fig. 1.15. The robot location and the corresponding sensory readings can be predicted in a future moment without actually driving the robot in the environment.

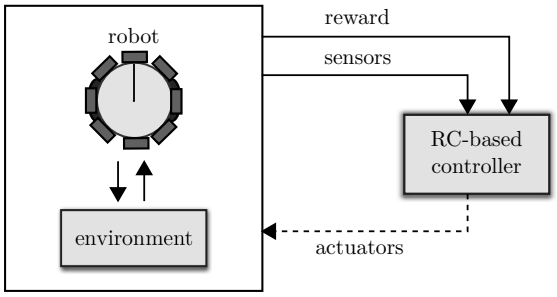


Figure 1.17: Reinforcement learning of environment-robot dynamics with RNNs. Dashed connections are trainable.

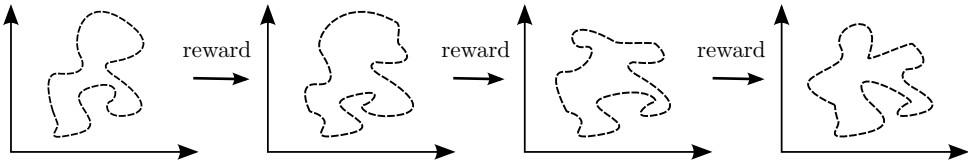


Figure 1.18: Reinforcement Learning shapes navigation attractor in bi-dimensional dynamical system space as learning evolves. The attractor is dynamic, i.e., changes over time with learning iterations.

navigation attractor made only by self-predictions. This is possible because the coupling results from the internal self-predictions, instead of coupling with the actual environment. Thus, this internally generated attractor only exists in the dynamical system space since the environment is not coupled. Fig. 1.16 shows an example of the navigation attractor in the dynamical system space and corresponding predictions of sensory nodes and context nodes which are fed back to the reservoir.

This predictive simulation of behaviors could be combined with the *trick* of shifting the dynamical system space to confined subspaces by the use of binary inputs when multiple behaviors should be modeled (as done in Section 1.3.2). In this way, these attractors in the high-dimensional reservoir space can be linearly discriminated, making possible the predictive simulation of multiple behaviors.

1.3.6 Iterative Shaping of Navigation Attractors

The aforementioned schemes of modeling navigation attractors with dynamical systems use supervised learning techniques. After one-

shot offline training (through linear regression on the reservoir states), the reservoir embeds the behaviors by observation of the supervisor controller. A second approach to learn navigation attractors is through reinforcement learning (RL). Under this scheme, the RC network does not receive a teacher signal, but only a reward signal usually indicating success or failure (Fig. 1.17). Thus, learning is achieved by trial and error, which means that a lot of random trials will take place in the beginning of the learning process. During this iterative learning procedure, the navigation attractor learned by the RNN is actually dynamic, i.e., changing over time (Fig. 1.18).

1.4 This thesis

The following sections present the main contributions and results of this work as well as the structure of this thesis.

1.4.1 Goals and Contributions

This thesis follows a dynamical systems' approach for modeling autonomous navigation systems for **small energy efficient mobile robots**. The goal is to achieve high-level intelligent navigation using a **low-cost sensory apparatus**, such as few **noisy distance sensors** of limited accuracy, to perceive the environment. With these aims, intelligent mobile robots becomes commercially viable as domestic products which can be used for efficient trajectory generation with inexpensive home cleaning robots such as the *iRobot family of robots*.

The dynamical systems' approach is implemented using **Reservoir Computing (RC) networks**, which enables powerful temporal processing simply using linear regression methods. The dynamical system (i.e., the reservoir) is an RNN with fixed weights used as a temporal kernel to *project the low-dimensional sensory input* (given by few noisy distance sensors) *into a high-dimensional nonlinear space*. Learning and prediction occurs at a separate output layer. This characteristic is the essential feature which enables the uncomplicated learning of navigation attractors (or behaviors) as well as abstract sensor-based environment representations using small robots with few noisy sensors.

The dynamical system space has an inherent fading memory which is able to **disambiguate the sensory space from sensor aliasing problems**. This work uses this property for efficient **prediction of the location of a mobile robot**, driven by a reactive behavior, in **unknown, dynamic, and even symmetric environments**. All of this is achieved **without the use of odometry information** from wheel encoders, so that the robot becomes situated in its environment solely based on the short-term memory from the dynamical reservoir of previous sensory inputs.

This work shows an innovative way to model **goal-directed navigation** from a bottom-up approach, by the combination of two RC networks in a hierarchical architecture, one which learns sensor-based spatial representations of an environment and another one which models reactive navigation attractors. Context switches, provoked by crossing doors connecting two rooms, are able to direct navigation as a sequence of robust reactive navigation behaviors.

A **first step towards path planning with RC networks** is given in this thesis. By using a generative RC architecture which trains sensory nodes as well as nodes used for location detection, the network can be autonomously simulated using its own predictions in closed loop mode, by shutting off the feedback from the environment after training. This technique when applied to multiple behaviors can be used to generate predictive trajectories for each reactive behavior, without actually moving the robot in the environment.

All of the aforementioned tasks are accomplished in a supervised learning setting, which can be useful, for example, when *teaching or programming behaviors* by demonstration. A contribution towards autonomy of the learning process in this thesis is given by the **unsupervised learning of sensor-based spatial representations** for small mobile robots in unstructured environments. In principle, this can make goal-directed navigation much more accessible due to non-requirement of labeling locations in the training data.

Finally, I show that **context-based navigation attractors can be iteratively shaped** through reinforcement learning, on the base of a sequential execution of simple motor primitives and without the need to show the desired behavior through a supervisor (Appendix A).

The research accomplished and reported in this thesis considers the following important points:

- the trained systems in this thesis make complex predictions receiving only **low-dimensional input** such as from a few distance sensors of a small mobile robot;
- no pre-processing (apart from normalization) is applied to the input signal (distance sensors). The dynamical reservoir system is excited by the **raw input**;
- robot **sensors are noisy**;
- the **environment of the robot is unknown** (not explicitly modeled);
- the robot becomes **situated** through the coupling of the dynamical system and the environment observed from the robot's point of view.

A list of publications is presented in a separate section at the end of this book, including the results from this doctorate research.

1.4.2 Structure

Each chapter is summarized below in order of appearance. The schematic overview showing the interdependence between chapters or subjects is given in Fig. 1.19.

Reservoir Computing (Chapter 2)

In this chapter, a detailed description on Reservoir Computing is given, specifically on Echo State Networks, an RC model for analog sigmoid reservoir units. Leaky-integrator units which provide reservoirs with an flexible way to tune their dynamics to slower frequencies are discussed here. Following that, two training methods for the output layer are presented: Least Squares method and Ridge Regression (Regularized Least Squares). The following section elaborates on how to use RC networks to perform dynamic pattern recognition, inclusive considering unbalanced datasets. Error measures, parameter optimization issues are also presented.

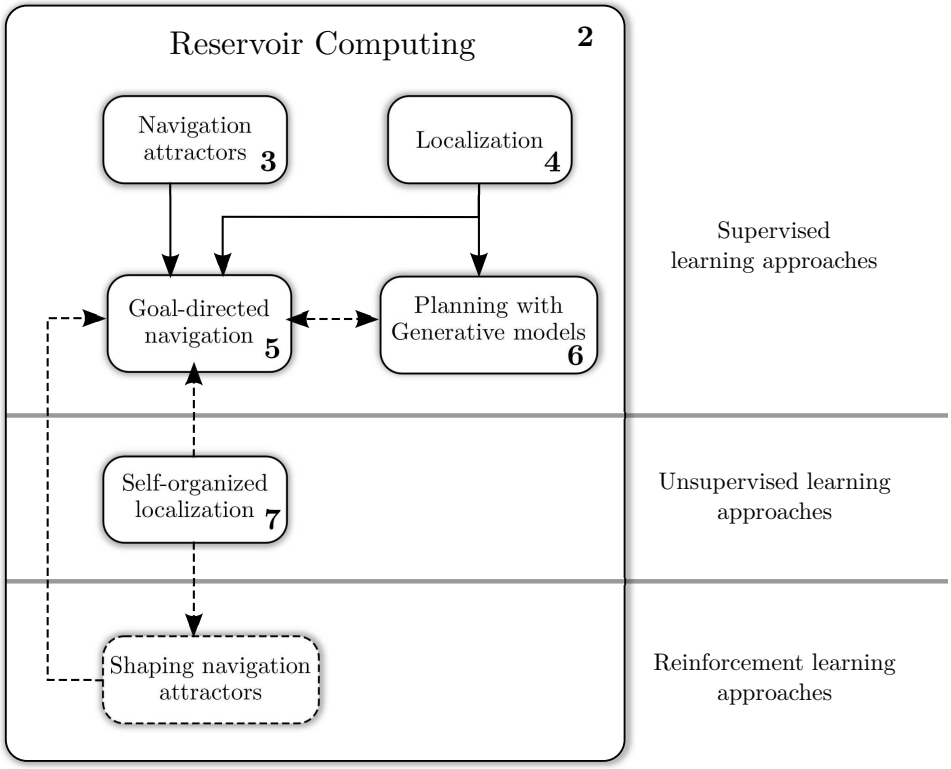


Figure 1.19: Schematic overview of this thesis showing the relationships between chapters/subjects given by solid arrows representing dependences implemented in this work and dashed arrows representing future possible extensions of this work. The dashed box on *Shaping Navigation attractors* is early work given as an Appendix in this thesis.

Supervised Learning of Navigation Behaviors (Chapter 3)

This chapter shows that an RC network can model multiple navigation attractors (or reactive behaviors) by supervised learning of the desired sensory-motor coupling, where each behavior is learned by shifting the high-dimensional space of the reservoir with additional binary inputs. A simulated robot becomes situated when performing the reactive behavior by the coupling of the dynamical system and environment. A second experiment models short-term memory navigation tasks, such as the T-maze task, where the mobile robot has to *remember* a previously given temporary stimulus for correct decision making at a future moment.

Robot Localization (Chapter 4)

Sensor-based spatial representations of several simulated and real environments and robot models are modeled in a supervised way with RC networks. As the dynamical reservoir has an inherent fading memory of past inputs, the system is able to detect the locations as the robot navigates by considering only the current local sensory input given by few noisy distance sensors, overcoming sensor aliasing problems. This ultimately leads to a dynamical system based robot localization system which does not require the use of odometry.

Goal-directed Navigation (Chapter 5)

This chapter builds upon previous results, by combining the modeling of navigation attractors of Chapter 3 and the localization capability of Chapter 4 for designing a goal-directed navigation system. The architecture is organized in hierarchies where the localization module provides context switches for selecting different behaviors in the navigation module. It is purely based on a supervised learning procedure where a supervisor generates examples of routes through multi-room environments using a physically realistic robot model.

Generative Modeling of Environment-Robot Dynamics (Chapter 6)

A generative extension of the localization architecture from Chapter 4 which also trains sensory nodes is presented in this chapter. The generative network is able to model reactive robot behaviors,

which is clearly verified when the internal network dynamics is simulated by shutting off the sensory feedback from the environment and letting the network autonomously generate what it has learned, leading to path planning applications.

Unsupervised Learning for Robot Localization (Chapter 7)

Chapter 7 tackles the self-organized formation of spatial representations of simulated and real-world environments using an hierarchical architecture and an existent unsupervised learning technique called Slow Feature Analysis (SFA). The linear SFA output layer learns to extract slowly-varying features from the non-linear reservoir state space, forming a non-localized environment representation. A final upper layer applies sparse coding on the SFA outputs, leading to the formation of a localized spatial representation similar to ones given by hippocampal CA1 cells of rats in W-tracks.

Conclusion and Future Work (Chapter 8)

Finally, conclusion and future work is given in Chapter 8.

Appendices

Appendix A describes the research and experimental results on *Reinforcement Learning in non-Markovian Navigation Tasks*. Appendix B presents all (simulated and real) robot models used in this work.

2
Reservoir Computing

2.1 Introduction

Traditional neural network models were designed to process static spatial input patterns, and are not inherently able to handle time-varying stimuli or dynamic patterns. To cope with temporal problems, these networks model time as an additional spatial dimension by dividing time into bins such that, for example, a 5-bin time window yields an input layer size of $5n_i$, where n_i is the number of input signals. In this way, time is treated as an additional spatial dimension at the level of the inputs, which is not a biologically plausible approach. Moreover, this time window approach results in a high number of parameters if larger memory is required.

In a second approach for representing time, neural network models with recurrent connections allowed for computation based on the previous state of the network and the current sensory input (Elman, 1990; Jordan, 1986), providing a mechanism of temporal context that still considered time as a discrete dimension (Buonomano and Maass, 2009).

The current work is based on the Reservoir Computing (RC) paradigm (Verstraeten et al., 2007), where a randomly generated non-linear dynamical system with fixed parameters (weights), such as an RNN, is used to map the inputs to a high-dimensional space, in which classification or linear regression is efficiently accomplished. The states of this dynamical system, called *reservoir*, are linearly combined in an adaptive readout output layer, which is the sole

trained part of the architecture. This type of *state-dependent computation* has been proposed as a biologically plausible model for cortical processing (Buonomano and Maass, 2009; Maass et al., 2002; Yamazaki and Tanaka, 2007). Such theoretical models include: *Echo State Networks* (Jaeger and Haas, 2004) for analog neurons and *Liquid State Machines* (Maass et al., 2002) for spiking neurons. Backpropagation-decorrelation (BPDC) (Steil, 2004) also uses analog sigmoidal neurons as reservoir, but the learning rule is attained from a different angle: it is based on an extension of the Atiya-Parlos recurrent learning (Atiya and Parlos, 2000) which restricts the adaptation of weights to the output layer.

From a machine learning perspective, a reservoir network, usually randomly generated and sparsely connected, functions as a *temporal kernel*, projecting the input to a dynamic non-linear space. During simulation, the reservoir states form a trajectory which is dependent on the current external sensory input, but which still contains memory traces of previous stimuli. Computation in the output layer occurs by linearly reading out instantaneous states of the reservoir. In this way, reservoir architectures can inherently process spatiotemporal patterns.

This chapter is devoted to introducing the Echo State Network (ESN) model, an RC architecture for analog networks which will be used for modeling several robotic tasks throughout this thesis. In the following, leaky integrator neurons for enhancing the reservoir's memory are presented, as well as the standard training method (e.g., Least Squares) for the readout output layer. Next, Ridge Regression and state noise injection are introduced as forms of regularization techniques against over-fitting. Classification with ESNs, error measures for regression and classification tasks, optimization of reservoir parameters are also presented in this chapter.

2.2 Echo State Network

An ESN is composed of a discrete hyperbolic-tangent RNN, the reservoir, and of a linear readout output layer which maps the reservoir states to the actual output. Let n_i, n_r, n_o represent the number of input, reservoir and output units, respectively, $\mathbf{u}[n]$ the

n_i-dimensional external input, $\mathbf{x}[n]$ the n_r-dimensional reservoir activation state, $\mathbf{y}[n]$ the n_o-dimensional output vector. Then the discrete time dynamics of the ESN is given by the state update equation

$$\mathbf{x}[n+1] = f\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[n] + \mathbf{W}^{\mathrm{r}}_{\mathrm{o}}\mathbf{y}[n] + \mathbf{W}^{\mathrm{r}}_{\mathrm{b}}\right), \tag{2.1}$$

where $f() = \tanh()$ is the hyperbolic tangent activation function, commonly used for ESNs, and by the output computed as:

$$\begin{aligned}\mathbf{y}[n+1] &= g\left(\mathbf{W}^{\mathrm{o}}_{\mathrm{r}}\mathbf{x}[n+1] + \mathbf{W}^{\mathrm{o}}_{\mathrm{i}}\mathbf{u}[n] + \mathbf{W}^{\mathrm{o}}_{\mathrm{o}}\mathbf{y}[n] + \mathbf{W}^{\mathrm{o}}_{\mathrm{b}}\right) && (2.2)\\ &= g\left(\mathbf{W}^{\mathrm{out}}\left(\mathbf{x}[n+1], \mathbf{u}[n], \mathbf{y}[n], 1\right)\right) && (2.3)\\ &= g\left(\mathbf{W}^{\mathrm{out}}\mathbf{z}[n+1]\right), && (2.4)\end{aligned}$$

where: g is a post-processing activation function; $\mathbf{W}^{\mathrm{out}}$ is the column-wise concatenation of $\mathbf{W}^{\mathrm{o}}_{\mathrm{r}}$, $\mathbf{W}^{\mathrm{o}}_{\mathrm{i}}$, $\mathbf{W}^{\mathrm{o}}_{\mathrm{o}}$ and $\mathbf{W}^{\mathrm{o}}_{\mathrm{b}}$; and $\mathbf{z}[n+1] = (\mathbf{x}[n+1], \mathbf{u}[n], \mathbf{y}[n], 1)$ is the extended reservoir state, i.e., the concatenation of the state, the previous input and output vectors and a bias term, respectively.

The matrices $\mathbf{W}^{\mathrm{to}}_{\mathrm{from}}$ represent the connection weights between the nodes of the complete network, where r, i, o, b denotes *reservoir*, *input*, *output*, and *bias*, respectively. All weight matrices representing the connections to the reservoir, denoted as $\mathbf{W}^{\mathrm{r}}_{\cdot}$, are initialized randomly (represented by solid arrows in Figure 1.4), whereas all connections to the output layer, denoted as $\mathbf{W}^{\mathrm{o}}_{\cdot}$, are trained (represented by dashed arrows in Figure 1.4). Fig. 2.1 is a schematic representation of Fig. 1.4 which shows the connections and the respective mappings given by the matrices $\mathbf{W}^{\cdot}_{\cdot}$ in (2.1) and (2.2).

Output feedback given by the projection $\mathbf{W}^{\mathrm{r}}_{\mathrm{o}}\mathbf{y}[n]$ and bias $\mathbf{W}_{\mathrm{b}}$ are optional. In the absence of these terms, (2.1) and (2.2) become:

$$\begin{aligned}\mathbf{x}[n+1] &= f\left(\mathbf{W}^{\mathrm{r}}_{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}^{\mathrm{r}}_{\mathrm{i}}\mathbf{u}[n]\right) && (2.5)\\ \mathbf{y}[n+1] &= g\left(\mathbf{W}_{\mathrm{o}}\mathbf{x}[n+1]\right). && (2.6)\end{aligned}$$

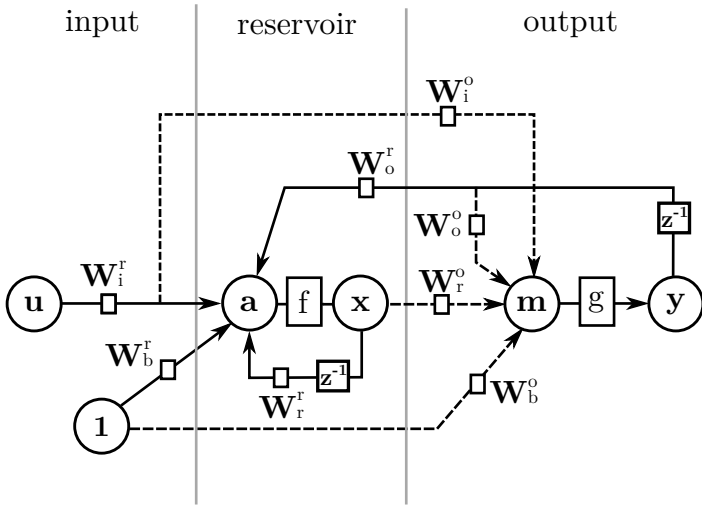


Figure 2.1: Reservoir Computing mapping scheme. Dashed connections are trainable whereas solid connections are fixed.

Table 2.1: Elements of Fig. 2.1

Signals	
$\mathbf{u}$	input signal
$\mathbf{y}$	output signal
$\mathbf{x}$	reservoir state
$\mathbf{a}$	weighted sum for reservoir units
$\mathbf{m}$	weighted sum for output units
Weights	
$\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$	input to reservoir connection matrix
$\mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$	bias to reservoir connection matrix
$\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$	reservoir connection matrix
$\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$	output to reservoir connection matrix
$\mathbf{W}_{\mathrm{i}}^{\mathrm{o}}$	input to output connection matrix
$\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$	reservoir to output connection matrix
$\mathbf{W}_{\mathrm{o}}^{\mathrm{o}}$	output to output connection matrix
$\mathbf{W}_{\mathrm{b}}^{\mathrm{o}}$	bias to output connection matrix

2.3 Reservoir Design and Dynamics

In this section, the procedures for reservoir creation and dynamics tuning are presented. The non-trainable connection matrices $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}, \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}, \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}, \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}$ are usually generated from a random distribution, such as a Gaussian distribution $N(0, 1)$ or a uniform discrete set $\{-1, 1\}$. During this initialization, two parameters are used:

the connection fraction $c_{\mathrm{from}}^{\mathrm{to}}$ corresponds to the percentage of nonzero weights in the respective connection matrix $\mathbf{W}_{\mathrm{from}}^{\mathrm{to}}$.

the scaling $\upsilon_{\mathrm{from}}^{\mathrm{to}}$ corresponds to the scaling of the respective connection matrix $\mathbf{W}_{\mathrm{from}}^{\mathrm{to}}$ such that all weights are rescaled according to multiplication $\upsilon_{\mathrm{from}}^{\mathrm{to}} \mathbf{W}_{\mathrm{from}}^{\mathrm{to}}$.

While the connectivity between units in $\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$ and $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ is not that important for **analog, rate-based reservoirs**[1], although they are usually created considering sparse connectivity for historic reasons, the **scaling** of these matrices has a great influence on the reservoir dynamics (Verstraeten et al., 2007) and must be tuned for optimal performance.

The randomly generated $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ must be rescaled such that the dynamical system is stable[2] but it still exhibits rich dynamics. As the ESN is usually nonlinear, this can be achieved by studying a linearized version of the ESN around the equilibrium point (Kuznetsov, 1998). Under this assumption, a necessary condition to guarantee the **Echo State Property** (ESP) (Jaeger, 2001) for ESNs, i.e., a reservoir with fading memory[3], is to rescale $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ such that the maximal singular value of $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ is smaller than unity. Conversely, the ESP is violated for zero input if the **spectral radius**[4] $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}})$ of the reservoir connection matrix $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ is larger than unity.

[1] While connectivity or topology structure is not important for analog networks, it is very important for binary (spiking) neurons (Schrauwen et al., 2008).

[2] Sometimes, $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ is rescaled such that few eigenvalues are situated slightly above the unity circle, which can offer better performance in some tasks.

[3] The Echo State Property states conditions for the ESN principle to work. It can be understood as having a reservoir with fading memory which asymptotically washes out any information from initial conditions.

[4] The spectral radius $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}})$ is the largest absolute eigenvalue of the reservoir connection matrix $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$.

However, using the maximal singular value to rescale the reservoir connection matrix usually does not provide rich reservoir dynamics. An alternative is to rescale $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ such that its spectral radius $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) < 1$ (Jaeger, 2001). Although it does not guarantee the ESP, it has been empirically observed that this criterium works well and often produces analog sigmoid ESNs with ESP for any input, producing richer reservoirs which contain signals with multiple frequencies. For most applications, the best performance is attained with a reservoir that operates at the **edge of stability**, e.g., $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.99$.

Besides the reservoir weight matrix, the dynamics of the reservoir is influenced by several factors: the input scaling υ_i^r of $\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$, an optional bias, the nonlinearity of the nodes and the external input driving the reservoir (Verstraeten and Schrauwen, 2009). Considering a normalized input signal $\mathbf{u}[n]$, the effect of input scaling υ_i^r on the reservoir dynamics is such that, the larger the scaling, the closer to saturation the reservoir states will be, since the reservoir state is shifted torwards the non-linear area of the tanh activation function. Spectral radius closer to unity as well as larger input scaling makes the reservoir more non-linear, which has a deterioration impact on the memory capacity as side-effect (Verstraeten et al., 2010). Moreover, the larger the input scaling, the further above unity the spectral radius might be while still attaining the ESP.

The scaling of these non-trainable weights is a parameter which should be chosen according to the task at hand empirically, analyzing the behavior of the reservoir state over time, or by grid searching over parameter ranges (see also Section 2.9).

An ESN without output feedback is inherently stable due to the ESP. However, with nonzero output feedback, stability can not be always guaranteed. A formal analysis of the stability of the ESN in this case is challenging. Nevertheless, stabilizing solutions include the use of regularization techniques such as the addition of state noise during training (Jaeger, 2002b) (see Section 2.6).

More recently, *effective spectral radius* (Ozturk et al., 2006) has been proposed as a dynamical measure of stability for ESNs, which results from the linearization of the dynamical system *around the current state*. Another dynamical measure is the *local Lyapunov exponent* introduced in Verstraeten and Schrauwen (2009).

2.4 Leaky-integrator Units and Timescales

Making the reservoir units leaky integrators offers possibilities for adjusting the timescale of the reservoir to the temporal dynamics of the input signal. This can be achieved after discretizing the following differential equation representing the reservoir in continuous time[5]

$$\dot{\mathbf{x}} = \frac{1}{c}\left(-a\mathbf{x} + f(\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u} + \mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x})\right), \tag{2.7}$$

where a is the leaking rate of the units and c is a scaling factor. Setting $a = 1$ and using the Euler method, it is possible to get

$$\mathbf{x}((t+1)\delta) = (1-\alpha)\mathbf{x}(t\delta) + \alpha f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}(t\delta) + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}(t\delta)\right), \tag{2.8}$$

where δ is the Euler stepsize and $\alpha = \delta/c$. Assuming the following change in notation $\mathbf{x}[n] = \mathbf{x}(t\delta)$, from continous to discrete time, then (2.8) becomes

$$\mathbf{x}[n+1] = (1-\alpha)\mathbf{x}[n] + \alpha f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}[n]\right) \tag{2.9}$$

which is represented schematically in Fig. 2.2(a). The equation above is equivalent to low-pass filtering of the reservoir states with a cut-off frequency $\alpha/(1-\alpha)$. Whereas (Jaeger et al., 2007) investigates the parameter a, this work uses α for fine-tuning the temporal dynamics of the reservoir similarly to Schrauwen et al. (2007). The advantage of using α over a is that the spectral radius of $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ does not change when α changes.

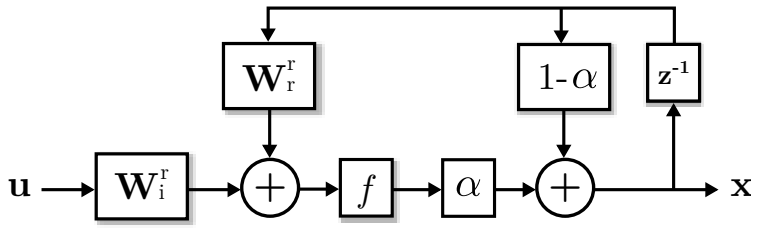
The loop of the integrator can be put over the nonlinearity, as shown in Fig. 2.2(b), in which case the reservoir state update equation is written as follows:

$$\mathbf{x}[n+1] = f\left((1-\alpha)\mathbf{x}[n] + \alpha(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}[n])\right). \tag{2.10}$$

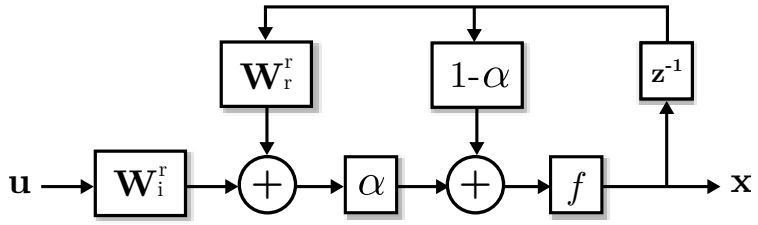
This latter form has the advantage of damping the activation of the reservoir state[6] due to placement of the nonlinearity over the loop.

[5] For the sake of simplicity, output feedback weights $\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$ and bias terms $\mathbf{W}_{\mathrm{b}}$ are not considered in the equations of leaky integrator units.

[6] This contracting property tend to shift the reservoir to the linear regime for small leak rates α (i.e., slow reservoirs), a similar effect achieved by small input



(a) Integrator after the nonlinearity



(b) Integrator over the nonlinearity

Figure 2.2: Reservoir schemes with leaky-integrator units, where the loop of the integrator is positioned after the nonlinearity (a) and over the nonlinearity (b).

Furthermore, it can be implemented without additional overhead, by redefining the matrices $\mathbf{W}_\mathrm{r}^\mathrm{r}$ and $\mathbf{W}_\mathrm{i}^\mathrm{r}$:

$$\tilde{\mathbf{W}}_\mathrm{r}^\mathrm{r} = \alpha \mathbf{W}_\mathrm{r}^\mathrm{r} + (1-\alpha)\mathbf{I}, \tag{2.11}$$

$$\tilde{\mathbf{W}}_\mathrm{i}^\mathrm{r} = \alpha \mathbf{W}_\mathrm{i}^\mathrm{r}. \tag{2.12}$$

Then, the reservoir state update equation becomes:

$$\mathbf{x}[n+1] = f\left(\tilde{\mathbf{W}}_\mathrm{r}^\mathrm{r}\mathbf{x}[n] + \tilde{\mathbf{W}}_\mathrm{i}^\mathrm{r}\mathbf{u}[n]\right). \tag{2.13}$$

There are two ways to increase the memory of a reservoir which has no output feedback. It is possible to either tune the leak rate $\alpha \in (0, 1]$ of the reservoir for matching the timescale of the input signal or downsample the input signal. Low leak rates yield reservoirs with more memory which can *remember* the previous stimuli for longer time spans. On the other hand, leak rates close to 1 are suitable for high-frequency input signals which vary in a faster timescale. Sometimes a combination of reservoir units possessing distinct leak rates can improve performance (Antonelo et al., 2008a), for example,

scaling.

when the input signal consists of components operating in distinct timescales. In this case, if each reservoir unit has its own leak rate, equations (2.9) and (2.10) becomes respectively:

$$\mathbf{x}[n+1] = (1-\vec{\alpha}) \odot \mathbf{x}[n] + \vec{\alpha} \odot f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}[n]\right) \quad (2.14)$$

$$\mathbf{x}[n+1] = f\left((1-\vec{\alpha}) \odot \mathbf{x}[n] + \vec{\alpha} \odot \left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}[n]\right)\right), \quad (2.15)$$

where $\vec{\alpha} = [\alpha_1 \alpha_2 \cdots \alpha_{n_r}]$ is the vector with each unit's leak rate; and $\odot$ is the notation for component-wise multiplication of vectors.

The **downsampling rate** d_t reduces the original input signal to n_s/d_t samples, where n_s is the total number of samples, generating a smoothed downsampled input signal[7], which has been shown to have a similar effect to using leak rates in the reservoir (Schrauwen et al., 2007). The advantage of downsampling the training data over using leak rates is that it requires less memory, as the data size is reduced. Sometimes, a combination of downsampling and leak rates gives the best performance.

Other approaches include the use of band-pass filters in reservoir units (wyffels et al., 2008) which allow for very specific frequency sensitivity.

2.5 Readout Output Training

The readout output of the RC network is the only layer to be trained, usually by standard **linear regression** methods. For that, the reservoir is driven by an input sequence $\mathbf{u}(1), \ldots, \mathbf{u}(n_s)$ which yields a sequence of extended reservoir states $\mathbf{z}(1), \ldots, \mathbf{z}(n_s)$ using (2.1).

The desired teacher outputs $\hat{\mathbf{y}}[n]$ are collected row-wise into a matrix $\hat{\mathbf{Y}}$. The generated extended states are collected row-wise into a matrix $\mathbf{X}$ of size $n_s \times (n_r + n_i + 1)$ if no output feedback is used. If the output is fed back to the reservoir via nonzero $\mathbf{W}_{\mathrm{o}}^{\mathrm{r}}$, then $\mathbf{X}$ has size $n_s \times (n_r + n_i + n_o + 1)$ and, during the generation of the extended states $\mathbf{z}(n_s)$, the desired output values $\hat{\mathbf{y}}[n]$ are written into

[7] The downsampling in this thesis is accomplished using the *resample* function in Matlab, which applies an anti-aliasing (lowpass) FIR filter to the input during the resampling process. This function is always used unless otherwise stated.

the output units through *teacher forcing*[8]. Then the training of the output layer consists of finding the weights $\mathbf{W}^{\text{out}}$ which minimizes the sum of squared errors

$$\sum_{t=1}^{n_s} \left(\hat{\mathbf{y}}[n] - \mathbf{y}[n]\right)^2, \tag{2.16}$$

by applying the *Wiener-Hopf* solution (or **Linear Least Squares**):

$$\mathbf{W}^{\text{out}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \hat{\mathbf{Y}} \tag{2.17}$$

where n_s denotes the total number of training samples and the initial state is $\mathbf{x}(0) = \mathbf{0}$. Note that the other matrices ($\mathbf{W}_{\text{r}}^{\text{r}}, \mathbf{W}_{\text{i}}^{\text{r}}, \mathbf{W}_{\text{b}}^{\text{r}}, \mathbf{W}_{\text{o}}^{\text{r}}$) are not trained at all.

It is important to note that there is an initial transient during the generation of reservoir states $\mathbf{x}[n]$ using (2.1) due to the fading memory of the reservoir, which may be undesired for the readout training. So, the usual procedure to deal with this is to disregard the first n_{wd} samples in a process called **warm-up drop** so that only the samples $\mathbf{z}[n], n = n_{wd}, n_{wd}+1, ..., n_s$ are collected into the matrix $\mathbf{X}$. Although this procedure is always used in this work, the notation for the generation of reservoir states will not change for the sake of simplicity.

The learning of the RC network is a fast process without local minima. Once trained, the resulting RC-based system can be used for real-time operation on moderate hardware since the computations are very fast (only matrix multiplications of small matrices).

2.6 Regularization

In order to avoid over-fitting, a regularization term λ can be added to the error function in (2.16) (Bishop, 2006) which will keep the

[8] Teacher forcing is a procedure which feeds the target output signal to the reservoir (via the output units) instead of the actual network output. This is done during training when feedback connections from the output to the reservoir exist, but can also be used after training.

weights small by penalizing weight vectors with a great norm:

$$\frac{1}{2}\left\|\hat{\mathbf{y}} - \mathbf{w}_{\mathrm{o}}^{\mathrm{T}}\mathbf{X}\right\|_2^2 + \frac{\lambda}{2}\left\|\mathbf{w}_{\mathrm{o}}\right\|_2^2, \tag{2.18}$$

where $\mathbf{z}[n]$ is the extended harvested reservoir state as in (2.4); $\mathbf{X}$ is the row-wise collection of vectors $\mathbf{z}[n]$; and $\mathbf{w}_{\mathrm{o}}$ is the weight vector of one output unit y. Setting the gradient with respect to $\mathbf{w}_{\mathrm{o}}$ of the above expression to zero, one obtains:

$$\tilde{\mathbf{W}}^{\mathrm{out}} = (\mathbf{X}^{\top}\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^{\top}\hat{\mathbf{Y}} \tag{2.19}$$

where $\tilde{\mathbf{W}}^{\mathrm{out}}$ is the column-wise concatenation of $\mathbf{W}_{\mathrm{r}}^{\mathrm{o}}$, $\mathbf{W}_{\mathrm{i}}^{\mathrm{o}}$, $\mathbf{W}_{\mathrm{o}}^{\mathrm{o}}$ (without the bias term $\mathbf{W}_{\mathrm{b}}^{\mathrm{o}}$). This method is called **Ridge Regression** in statistics (Bishop, 2006), *Regularized Linear Least Squares* or *Tikhonov regularization* (Tychonoff and Arsenin, 1977). A second form of regularization is to add white **noise to the state update equation** during the generation of matrix $\mathbf{X}$ with the extended reservoir states $\mathbf{z}[n]$ (Jaeger, 2002a). In this case, (2.1) becomes during training:

$$\mathbf{x}[n+1] = f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[n] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\mathbf{u}[n] + \mathbf{W}_{\mathrm{o}}^{\mathrm{r}}\mathbf{y}[n] + \mathbf{W}_{\mathrm{b}}^{\mathrm{r}}\right) + \nu[n], \tag{2.20}$$

where $\nu[n]$ is a noise vector, where the noise variance σ_{ν}^2 is a parameter to be tuned. The noise injection term $\nu[n]$ may sometimes appear inside the nonlinearity f.

Both forms of regularization require the optimization of the parameters λ or σ_{ν}^2 so that the test error is minimum. This is usually done using cross-validation.

2.7 Classification and Fisher relabeling

Not only regression tasks, but also classification problems can be handled with reservoir computing networks. As the temporal aspect is inherent to RC networks, they can be used for **dynamic pattern recognition** (Jaeger, 2002a), where the patterns to be detected are extended over a short time interval.

In this work, **multi-label classification** is often used, for instance, in event and location detection for mobile robots (see Chap-

ter 4), where a temporally extended input pattern (distance sensor readings) can be classified as belonging to one class among several possible ones (events or robot locations).

The classification task is defined as follows. Let $\mathbf{y}[n]$ be the n_o-dimensional output vector of the RC network, where n_o is also the number of patterns to be detected. In an one-versus-all classifier, the desired output $\hat{\mathbf{y}}[n]$ is built over time such that, for each timestep, only one dimension is $+1$ whereas the others are -1, i.e., $\hat{\mathbf{y}}[n] = (-1, -1, \cdots, 1, \cdots, -1)_{1\times n_o}$. This can be extended to a situation where *no pattern is detected* by setting all components of the desired output to -1, that is, $\hat{\mathbf{y}}[n] = (-1, -1, \cdots, -1)_{1\times n_o}$. Then, the desired output vector $\hat{\mathbf{y}}[n]$ is collected row-wise into a matrix $\hat{\mathbf{Y}}_{n_s\times n_o}$ for $n = 1, 2, \cdots, n_s$. Training proceeds as described in Section 2.5 or Section 2.6 using the regularized least squares. After finding $\mathbf{W}^{\text{out}}$, in the test phase, the network output $\mathbf{y}[n]$ is post-processed by a winner-takes-all activation function:

$$g(y_i[n]) = \begin{cases} +1, & \text{if } y_i[n] > y_j[n], \forall j \neq i \\ -1, & \text{otherwise} \end{cases} \tag{2.21}$$

and in the case where patterns can be absent for any given time interval (i.e., event detection), the post-processing function becomes:

$$g(y_i[n]) = \begin{cases} +1, & \text{if } y_i[n] > \theta \text{ and } y_i[n] > y_j[n], \forall j \neq i \\ -1, & \text{otherwise} \end{cases} \tag{2.22}$$

where θ is a given threshold constant.

It is very common that **datasets** are **unbalanced** regarding the number of samples of each class. These class inequalities negatively influence the linear regression training. While it may be undoable to modify the underlying sampling mechanism, it is possible to rebalance the original dataset by **relabeling the teacher output** $y_i[n]$ of i^{th} class as follows.

Let

$$P_i = |\{y_i[n] \mid y_i[n] = +1, n = 1, \cdots, n_s\}|$$

be the number of samples of class i with $+1$ output label and

$$N_i = |\{y_i[n] \mid y_i[n] = -1, n = 1, \cdots, n_s\}|$$

the number of samples of class i with -1 output label. Then, the desired output for each class i for all samples $n = 1, 2, \cdots, n_s$ are relabeled as follows:

$$\hat{y}_i^{\text{new}}[n] = \begin{cases} \frac{P_i+N_i}{P_i} & \text{if } \hat{y}_i[n] = +1 \\ -\frac{P_i+N_i}{N_i} & \text{if } \hat{y}_i[n] = -1 \end{cases}. \tag{2.23}$$

And for tasks like *event detection* where patterns may be absent for a given time interval, that is, $\exists n, \forall i : \; y_i[n] = -1$:

$$\hat{y}_i^{\text{new}}[n] = \begin{cases} \frac{\sum_k P_k}{P_i} & \text{if } \hat{y}_i[n] = +1 \\ -\frac{\sum_k P_k}{\sum_{k \neq i} P_k} & \text{if } \hat{y}_i[n] = -1 \end{cases}. \tag{2.24}$$

This re-weighting of categorical labels for the least squares training is known as **Fisher relabeling** (Duda et al., 2001). It effectively corrects the undesired shift of the classification hyperplane given by the linear regression due to the unbalanced dataset.

2.8 Error Measures

For **regression tasks**, the Normalized Mean Square Error (NMSE) is used as a performance measure and is defined as:

$$\text{NMSE} = \frac{\langle (\hat{y}[n] - y[n])^2 \rangle}{\sigma^2_{\hat{y}[n]}}, \tag{2.25}$$

where the numerator is the mean squared error of the output $y[n]$ and the denominator is the variance of desired output $\hat{y}[n]$.

In **multi-label classification** problems such as *robot location detection*, the performance is measured by the mean classification rate:

$$\text{CR} = \frac{|P|}{n_s} \tag{2.26}$$

where $P = \{\mathbf{y}[n] | n = 1, 2, \cdots, n_s \text{ and } \mathbf{y}[n] = \hat{\mathbf{y}}[n]\}$ is the set containing all correctly classified label samples.

In problems where for a long time no dynamic pattern is present,

like in *event detection* during robot navigation, it is interesting to also measure the performance rate (true positives rate) for each class or event i:

$$\mathrm{TP_i} = \frac{|\mathrm{A}_i|}{|\mathrm{B}_i|} \tag{2.27}$$

where

$$\mathrm{A}_i = \{\hat{y}_i[n]|(\hat{y}_i[n] = +1) \wedge (y_i[n] = \hat{y}_i[n]]), n = 1, 2, \cdots, n_s\}$$

is the set of correctly classified samples of class i and

$$\mathrm{B}_i = \{\hat{y}_i[n]|(\hat{y}_i[n] = +1), n = 1, 2, \cdots, n_s\}$$

is the set of samples with class i.

2.9 Parameter Tuning

Some parameters of the ESN model do not require tuning, while others require optimization to get improved classification or regression performance on a test set.

Reservoir size n_r refers to the model capacity. Test performance increases asymptotically with the reservoir size if a regularization method is used. Larger reservoirs can solve more complex modeling problems which require richer nonlinear transformations of the input as well as provide longer-term fading memory to hold past inputs.

Spectral radius $\rho(\mathbf{W}_\mathrm{r}^\mathrm{r})$ Typically setting the reservoir at the edge of stability, i.e., $\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) = 0.99$ works very well for most scenarios. To achieve a more linear reservoir and, consequently, more memory capacity, a smaller spectral radius could be used.

Connectivity c: The connectivity between neurons in the reservoir and from input and output to reservoir is of less importance, as it does not affect performance for analog reservoirs such as an ESN (not the case for spiking networks) (Buesing et al., 2010).

Leak rate α is a parameter which needs tuning. The assumption is that the timescale in the reservoir should match the timescale of the task being modeled and, for that, the leak rate can be adjusted for optimal timescale matching.

Input downsampling d_t should be optimized for the same reasons as the leak rate. Sometimes, *input upsampling* may be desired for generating a longer input signal.

Input scaling υ_i^r also changes the reservoir dynamics and consequently the task performance. Input signals with a high amplitude will saturate the reservoir states, shifting the reservoir state to the non-linear regime of the tanh activation function f. Whereas this makes the reservoir more non-linear, it has a deterioration impact on the memory capacity (Verstraeten et al., 2010).

Output feedback scaling υ_o^r should be optimized for the generation task. This can cause dynamical instability since the ESP does not hold anymore when output feedback is used. Regularization should be used as way to stabilize the ESN.

Regularization parameter λ and noise variance σ_ν^2 If a non-regularized Least Squares method is used for training the readout, then the reservoir size should be optimized once a relatively *big* reservoir is subject to over-fitting. On the other hand, if *Ridge Regression* or *state noise injection* is used, the **regularization parameter** λ or the **noise variance** σ_ν^2 must be optimized. With proper regularization, performance improves asymptotically with the reservoir size.

These parameters, except for the first three, are usually optimized via **grid-search** with **k-fold cross-validation**. Other alternatives are to individually optimize the parameters with k-fold cross-validation or empirically hand-tune the parameters. In the former case, if one parameter affects the other, i.e., if they are inter-dependent, the solution is not optimal. Nevertheless, RC-based systems are somewhat robust to a diverse set of parameter ranges, including the random generation of matrices $\mathbf{W}_\cdot^{\mathrm{r}}$ (however, the rescaling of these matrices is important).

Table 2.2: Tips for Reservoir Design and Training

Parameter	Description	Suggested range/value
c_i^r	Input connection fraction	$[0.05, 0.5]$
v_i^r	Input scaling	**optimize**
d_t	Input downsampling	**optimize**
c_b^r	Bias connection fraction	$[0.05, 0.5]$
v_b^r	Bias scaling	**optimize**
n_r	Reservoir size	(the larger, the better)
c_r^r	Reservoir connection fraction	[0.01,1] (default to 1)
$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}})$	Spectral radius	0.99 (could be optimized)
α	Leak rate	**optimize**
c_o^r	Output feedback connection fraction	[0.05,0.5]
v_o^r	Output feedback scaling	**optimize**
σ_ν^2	Variance for state noise injection	**optimize**
λ	Regularization parameter	**optimize**

Table 2.2 shows a list of parameters related to reservoir design and training with corresponding suggested values or ranges. *Optimize* means that the parameter can be found by *trial and error* or by some automated method.

Although it is suggested that many parameters should be optimized, RC is quite robust to several of these parameters. Thus, it is relevant to mention the *three most important parameters for tuning*: **leak rate** (or, alternatively, resampling rate of the input signal), **input scaling**, and the **regularization parameter** (or, alternatively, variance for state noise). The other parameters are less important.

2.10 Conclusion

In this chapter, a detailed description of the Echo State Network model has been given. As the recurrent part of the network, i.e., the reservoir, is not trained, but only a readout output layer by standard linear regression methods, it constitutes an efficient model for recurrent network training. It overcomes the known convergence

problems, issues with vanishing gradients, and bifurcations existent in previous methods such as back-propagation through time. It was also shown that the reservoir can have its temporal dynamics tuned by making the units leaky-integrators, which is equivalent to low-pass filtering of the reservoir state.

Furthermore, the dynamic regime of the reservoir should be situated at the edge of stability so that interesting, rich dynamical reservoir activity is achieved. For that, the spectral radius of the connection matrix for the recurrent reservoir, the input scaling, and an optional bias are the main parameters which influence the dynamical behavior of the reservoir and should be tuned according to the task at hand.

Supervised training methods such as Least Squares and Ridge Regression were presented and classification with ESNs has been described. Additionally, error or performance measures for regression and classification tasks as well as parameter optimization issues have been presented in this chapter.

3
Supervised Learning of Navigation Behaviors

The first approach for modeling autonomous navigation systems for small mobile robots in this thesis is by supervised learning of robust reactive behaviors. By taking this approach, learning is accomplished by generating examples of the desired sensory-motor coupling using a supervisor.

3.1 Introduction

This chapter aims at automating the *programming* of simple reactive behaviors for small mobile robots such as the iRobot family of domestic products, which possess inexpensive sensory apparatus and represent a large potential market in the field of service robotics. To achieve that, a human instructor would show to the robot how to execute tasks, for instance, by giving examples of movements, behaviors or trajectories to be followed. The main idea is that a mobile robot should learn and generalize what it has learned according to a supervised learning process.

While similar work uses backpropagation through time (BPTT) to train RNNs for robot navigation problems (Tani, 1996; Tani and Nolfi, 1999) and for generating movements in a robotic arm (Tani, 2003), in this chapter Reservoir Computing (RC) networks are used as an efficient way for learning navigation behaviors by demonstration (e.g., with examples). The goal is to model a set of different sensory-motor couplings (or behaviors) using a single dynamical reservoir so that behaviors are represented in a distributed way in

the network. After the learning process, the coupling of the dynamical system (reservoir), which controls the robot, and the environment allows that the robot becomes situated in its environment since the internal state of the reservoir reflects the contextual state of the environment. This results from the direct relationship between the navigation behavior in the environment space (e.g., given by the robot trajectory) and the corresponding *sub-space attractor* in the dynamical system space, as it will be seen later in this chapter.

The first experiment deals with learning multiple conflicting behaviors, originated from different supervisor controllers, which can be switched by changing the operating point of the dynamical reservoir using an extra binary input channel (or vector, for more than two behaviors). It is shown that the RC network can model two navigation attractors and it is able to generalize these behaviors to bigger environments. The second experiment tackles partially observable tasks such as the T-maze task, which consists of a robot in a T-shaped environment that must reach the correct goal (left or right arm of the T-maze) depending on a previously received input sign. It is a control task in which the delay period between the sign received and the required response (e.g., turn right or left) is a crucial factor. Delayed response tasks like this one form a temporal problem that can be handled very well by RC networks, but considering a limited delay due to the reservoir's fading memory in networks with no output feedback.

3.2 Modeling Multiple Navigation Behaviors

In this section, two different nonlinear navigation behaviors are modeled by a single reservoir computing network which embeds these behaviors in sub-space attractors in the dynamical system space. An extra input channel to the network selects one of the navigation attractors, by shifting the operation point of the recurrent network into different sub-spaces. The behaviors are learned in an initial environment based on trajectories given by different teacher controllers, and are successfully used in a test environment not seen during training.

3.2.1 SINAR Robot Model

The following experiments are based on a robot model that is part of the 2D SINAR simulator (Antonelo et al., 2006) (this section is also presented in Appendix B. Its simulation environment generates the data necessary for training the RC networks. In SINAR, the mobile robot (Fig. 3.1) interacts with the environment by distance and color sensors; and by one actuator which controls the movement direction (turning). The environment of the robot is composed of several objects, each one of a particular color. Particularly, obstacles are represented by blue objects whereas targets are given by yellow objects. The robot model has 17 sensor positions distributed uniformly over the front of the robot, from -90° to +90°. Each position holds two virtual sensors for distance and color perception. The distance sensors are limited in range such that they saturate for distances greater than 300 distance units ($d.u.$), and are noisy - they exhibit Gaussian noise $N(0, 0.01)$ on their readings. A value of 0 means near some object and a value of 1 means far or nothing detected. At each iteration the robot is able to execute a direction adjustment to the left or to the right in the range [0, 15] degrees and the speed is equal to 0.28 distance units $(d.u.)/s$ (summary in Table 3.1).

SINAR Controller

The controller for the SINAR robot model (based on Antonelo et al. (2006)) is a complex intelligent navigation system composed of hierarchical neural networks which learn by classical reinforcement learning algorithms. The system learns to seek targets and avoid obstacles as the robot interacts with the environment, by colliding against obstacles and by capturing targets in the environment. It also learns to distinguish targets and obstacles by associating their distinct colors to attraction or repulsion behaviors (see Antonelo et al. (2005, 2006)). From now on, the controllers obtained from this model will be called INASY (Intelligent autonomous NAvigation SYstem). It is relevant to say that other controllers or supervisors could be used to generate training data or even possibly manually controlled data.

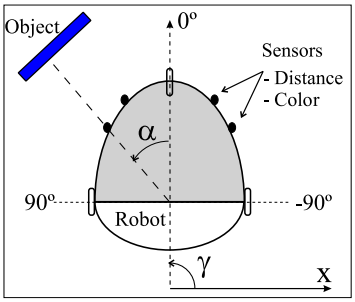


Figure 3.1: SINAR robot model.

Table 3.1: SINAR robot model

No. Dist. Sensors	17
No. Color Sensors	17
Range of Dist. Sens.	$300d.u.$
Noise on sensors	$N(0, 3d.u.)$
Speed	$0.28d.u.$/timestep

3.2.2 Training the Reservoir Architecture with Examples

The INASY controllers, described in the previous section, are teacher controllers responsible for providing examples of navigation trajectories to a RC-based robot controller which will be called RECNA (REservoir Computing NAvigation system) from now on.

The samples generated by INASY controllers containing data from distance and color sensors, and from actuators are used to train the RECNA controller in a Matlab environment using the RCT Toolbox[1] (Verstraeten et al., 2007). The experimental setup is given in the following section.

The supervised learning process uses training data from two different teacher INASY controllers. Consider that the data such as robot sensors and actuators obtained from both controllers are concatenated into a single dataset and that the total number of time samples is n_s. Then, training is performed using linear regression on the reservoir states as described in Section 2.5. Fig. 3.2 shows the data samples generated by different behaviors or teacher controllers,

[1]This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

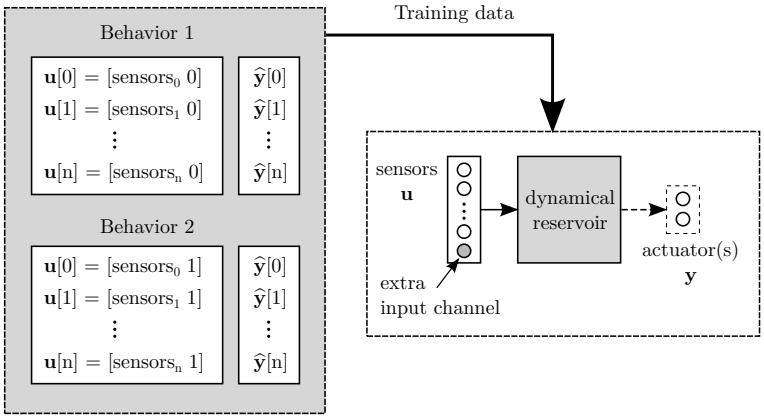
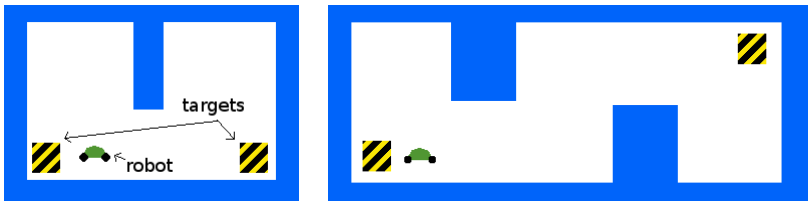


Figure 3.2: Training a single RC network for learning 2 different behaviors. Behaviors 1 and 2 are generated by distinct teacher controllers. The input $\mathbf{u}$ is the concatenation of the sensors and an extra input channel (0 or 1) for behavior selection.

which are used to train a single RC network that possess an extra input channel for behavior selection.

The supervised learning procedure can be summarized in four stages:

- First, the teacher INASY controllers navigate in a particular environment, e.g., avoid obstacles and/or seek targets.
- In a second stage, data samples with the observed sensory-motor couplings are collected from the INASY controllers during a robot run of a specific duration.
- If there are multiple behaviors possibly from different controllers, the third stage concatenates the data collected in the previous stage, and adds extra binary input channel(s) for behavior selection (where each possible binary value could correspond to a behavior, e.g., 01, 10, and 11 encode three different behaviors).
- The fourth stage corresponds to training the RECNA controller with the data collected in the second stage and concatenated in the third stage by supervised learning methods such as linear regression (Section 2.5).



(a) S1 (b) S2

Figure 3.3: Environments used for the experiments in this section. Initially, both targets are visible. After the robot captures one target, the other target is put back to its original location, making at least one target always visible. (a) Small environment with two targets and one robot. (b) Big environment with two targets and a robot.

3.2.3 Experiments

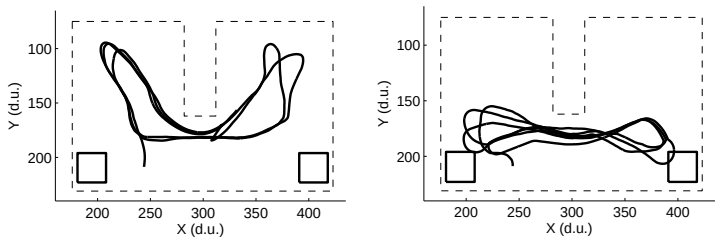
In this section, an RC network is trained to reproduce the following combined robot behaviors: **Environment Exploration** (EE) and **Target Seeking** (TS). The EE behavior makes the robot explore the environment but ignoring the targets, while the TS behavior makes the robot seek and capture targets in the environment as well as avoid obstacles.

The environments used for the experiments are shown in Fig. 3.3. The first environment is composed of a (blue) corridor with two (yellow) targets (the targets are striped in the figure for clarification). During simulation, the robot navigates through the environment normally performing cyclic trajectories. Captured targets are sequentially put back in the same locations after a capture[2]. Fig. 3.4 shows examples of navigation trajectories.

As EE and TS behaviors are conflicting behaviors, they must be generated by different INASY controllers. In the following, it is explained how these controllers are constructed using the intelligent navigation system described in Antonelo et al. (2006).

EE exploratory behavior The INASY controller which implements the EE behavior is trained to **avoid** blue objects (obstacles) and yellow objects (targets). An example of exploratory behavior which **ignores targets** is given in Fig. 3.4(a).

[2]A target capture causes the removal of the respective target from the environment.



(a) EE (b) TS

Figure 3.4: Example of navigation trajectories of teacher INASY controllers in environment S1. (a) EE exploratory behavior (ignores visible targets). (b) TS target seeking behavior (continually captures targets).

TS target seeking behavior The INASY controller that generates the TS behavior is trained to **avoid** blue objects (obstacles) and to **seek** yellow objects (targets). The resulting target seeking behavior is shown in Fig. 3.4(b).

Next, the samples with sensory and actuator information are collected from INASY controllers in two stages. In the **first stage**, the controller implementing EE behavior steers the robot in environment S1 from Fig. 3.3, exploring the environment and ignoring targets. All sensory inputs and actuators are recorded. In the **second stage**, the controller with TS behavior steers the robot in the same environment, but now generating a different trajectory towards the targets. Each stage lasts 22.500 timesteps, summing up 45.000 timesteps in total which corresponds approximately to 24 cyclic trajectories or loops in the respective environment.

After collecting the training data which represent EE and TS behaviors individually, a single RC network is trained to reproduce both behaviors by means of concatenation of the training data as well as of an extra input channel added for behavior selection, as described in previous section and in Fig. 3.2. If this extra input has value zero (one), then the EE (TS) behavior is selected.

3.2.4 Settings

The parameter configuration for the RC network of the RECNA controller is shown in Table 3.2. The inputs to the network are 17 distance sensors, 17 color sensors, plus 1 input for behavior selection,

Table 3.2: Parameter configuration for RECNA controller

Number of input channels	$n_i = 35$
Input connection fraction	$c_i^r = 0.2$
Input scaling	$v_i^r = 0.2$
Input downsampling	$d_t = 1$
Input to output connections	yes
Bias connection fraction	$c_b^r = 0.2$
Bias scaling	$v_b^r = 0.2$
Reservoir size	$n_r = 600$
Reservoir connection fraction	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 1$
Number of output channels	$n_o = 1$
Output feedback to reservoir	no

summing up $n_i = 35$ inputs. The reservoir size is $n_r = 600$ neurons. The output unit corresponds to the turning or direction adjustment robot actuator (the robot has constant velocity). The training is done according to Section 2.5 using the collected data of 45.000 timesteps, of which half of the observations has the value of the extra input channel set to 0 for EE behavior, and the other half has this value set to 1 for TS behavior.

3.2.5 Results

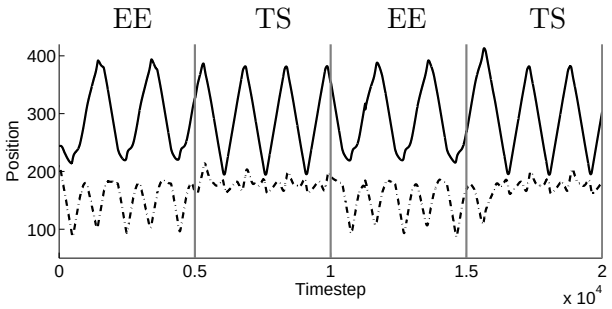
After learning in environment S1, the RECNA controller was evaluated in environments S1 and S2. The results for environment S1 are shown in Fig. 3.5. The simulation takes 20.000 timesteps. At each period of 5.000 timesteps, a behavior switching event takes place. Note that every switching implies a waiting time of 15 timesteps during which the robot is kept still so that a short reservoir transient is disregarded. After this switching interval, the reservoir is ready to drive the robot according to the selected behavior. Fig. 3.5(a) shows the coordinates of the robot during the run, where vertical lines represent the moments in which a behavior switching occurs. It can be seen that the behaviors are very well defined in their respective time interval. The trajectory of the robot changes as soon as the switching occurs and a target is localized. Fig. 3.5(b) shows

the corresponding robot trajectory in a 2D map during the simulation. The black (gray) trajectory corresponds to the time interval in which the EE (TS) behavior is selected.

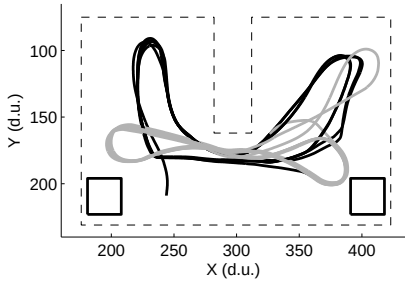
From these figures, it can be observed that the trajectories form navigation attractors in the environment. In addition, switching between these attractors is accomplished smoothly by the RECNA controller without collisions to obstacles.

Fig. 3.6 shows six randomly selected states from the reservoir over time as the RECNA controller drives the robot in environment S1. It is possible to observe that the dynamics of the reservoir changes at each moment of behavior switching given by the vertical lines in the figure. By reducing the high-dimensional state space of the dynamical reservoir, using Principal Component Analysis (PCA) on the reservoir states, it is possible to observe that *sub-space attractors* which are linearly separable (Fig. 3.7). By only changing an input from 0 to 1 or vice-versa, the operating point of the dynamical reservoir is changed to a different *sub-space attractor* in the dynamical system space, defined by the tight coupling between robot controller and environment, as introduced in Chapter 1.

Table 3.3 shows results for different number of neurons (n_r) in the reservoir. Each row shows the mean values of the: training NMSE error (defined in (2.25)), training time, number of target captures and number of collisions, considering 5 robot runs each of 20.000 timesteps and with a different randomly generated reservoir $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$. The training time includes the time to generate the matrix $\mathbf{X}$ and to compute (2.17) using an Intel Core2 Duo processor-based system. During a robot run, there are three switching events as in Fig. 3.5. The last column of the table presents the percentage of successful runs which have resulted in correctly performing the selected behaviors for all three events of behavior switching during the respective simulation. It can be observed that as the reservoir has more units, the performance of the resulting RECNA controller increases, e.g., by decreasing the number of collisions, although the training time also increases. For reservoirs containing more than 400 neurons, the resulting RC-based controllers are always stable, i.e., the selected task (EE or TS) is performed reliably. With a proper initialization of the reservoir weights, even small reservoirs with 100 units can perform these navigation tasks very well. As this small



(a)



(b)

Figure 3.5: Results for environment S1. (a) The coordinates of the robot are shown for 20.000 timesteps during the test phase. The solid and dashed lines are the x and y coordinates, respectively. Vertical gray lines represent the moments of behavior switching. (b) The corresponding trajectory of the robot in the Cartesian map. The solid black (gray) line represents the timesteps in which the selected behavior is the EE (TS) behavior.

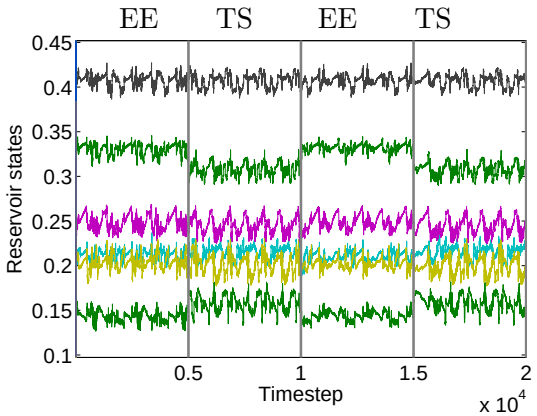


Figure 3.6: Reservoir states for the RECNA controller in environment S1. The plot shows six randomly chosen states from the reservoir. Vertical lines represent the moments in which the behavior switches.

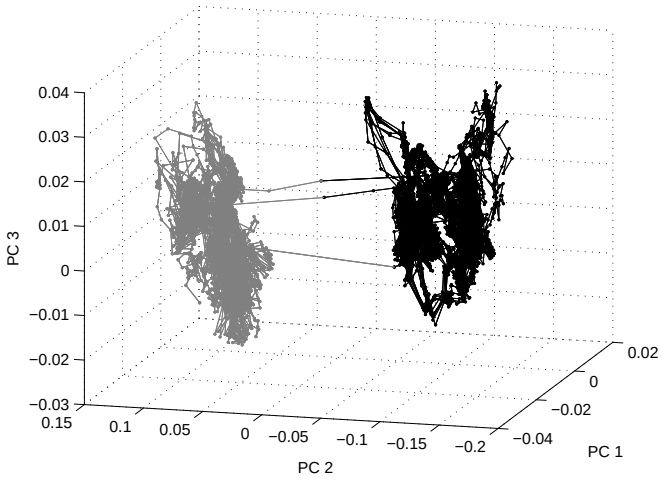


Figure 3.7: Three principal components of the reduced dynamical system state space after applying PCA on the reservoir states during testing with the RECNA controller in environment S1. Gray and black lines represent trajectories associated with different selected behaviors. The input channel for behavior selection effectively shifts the operating point of the reservoir state space into different linearly separable sub-space *attractors*. There are three switching events, represented by the lines connecting both sub-space attractors, as in Fig. 3.6. This figure is analogous to the fictitious example of Fig. 1.11.

Table 3.3: Mean results for different size of reservoirs - Environment S1

No. Neurons (n_r)	Training NMSE	Training Time (s)	No. Target Captures	No. Collisions	Correct behavior
100	0.88	5	12	20.6	40 %
200	0.85	9	12.2	11	80 %
400	0.82	25	11.8	0.8	100 %
600	0.80	60	12.6	0.6	100 %

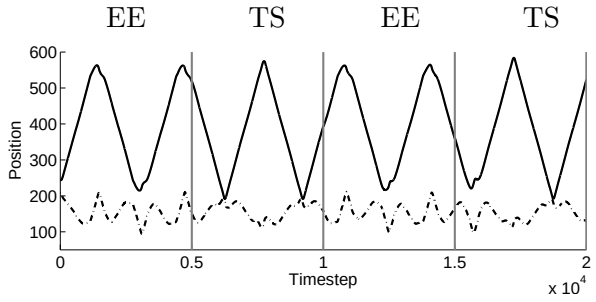
reservoir must be randomly generated, this *proper* initialization is obtained by generating reservoirs and testing the resulting controller until one solves the required task[3].

In order to test the generalization capabilities of the RECNA controller, a new environment S2 is considered which is different from the training environment (S1). The new environment (Fig. 3.3) is larger than S1, and has two targets, one located in the lower-left of the environment and another in the upper-right of the environment. The results in Fig. 3.8 show that the RECNA controller generalizes very well, being able to explore the environment when EE behavior is selected as well as to capture targets when TS behavior is chosen.

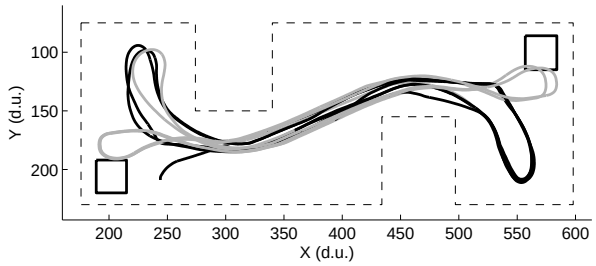
It is also interesting to compare the output of the teacher INASY controller with the output of the RECNA imitator controller. Fig. 3.9 shows that the output of the RC network is much less noisy than the output of the teacher controller, which yields a conclusion that the RC network stabilizes the sensory-motor coupling while disregarding the noisy superfluous signals from the teacher controller. The resulting signal smoothing is probably due to the fading memory property of reservoirs. In (Antonelo et al., 2008b), it is shown that RECNA controllers produce more stable behaviors than their teacher controllers. Experiments not shown in this work indicate that the trained RC-based controller is robust against noise and perturbations such as artificially pushing the robot around the environment in real time.

As a matter of comparison, instead of using RC networks, feed-forward networks such as the Multi-Layer Perceptrons (MLP) have

[3]On average, smaller randomly generated reservoirs have a lower probability of achieving a good performance and stable behavior than large reservoirs.



(a)

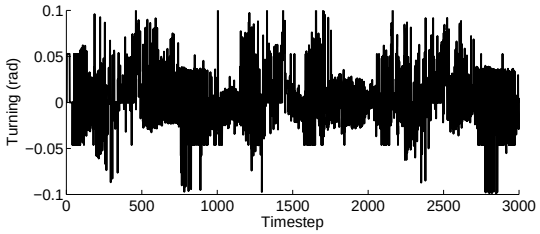


(b)

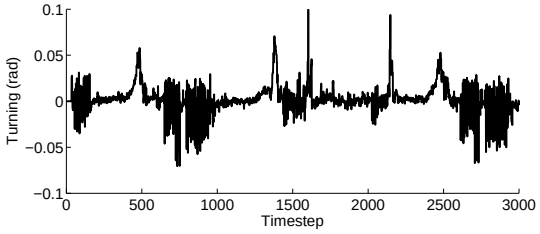
Figure 3.8: Results for generalization capabilities in environment S2. (a) The coordinates of the robot are shown for 20.000 timesteps during the testing. The solid and dashed lines are the x and y coordinates, respectively. Vertical gray lines represent the moment of behavior switching. (b) The corresponding trajectory of the robot in the Cartesian map. The solid black (gray) line represents the timesteps in which the selected behavior is the EE (TS) behavior.

also been used to model multiple robot behaviors using supervised learning. The MLP[4] was trained using the back-propagation algorithm to reproduce the same EE and TS behaviors given in Fig. 3.4. Although a different number of hidden layers have been tried, the MLP failed in all of them to drive the robot stably and safely (it constantly made the robot bump to the walls). Thus, *memoryless architectures* are not able to correctly model multiple behaviors and coordinate their switching.

[4]The experiment with the MLP does not use a time-window approach, making it a memoryless system.



(a) INASY



(b) RECNA (RC-based)

Figure 3.9: Outputs (turning actuators) of controllers for EE behavior during 3.000 timesteps. (a) Output from INASY controller. (b) Output from RECNA (the RC-based) controller.

3.3 Delayed Response Task: The T-maze

The road sign problem, which is tackled in this section, constitutes a particular temporal task which is defined in (Ulbricht, 1996). In this problem, an artificial agent (robot) which is driving along a corridor receives a temporary sign that must be remembered for future correct decision making. The T-maze task is the most common form of such problem: the robot drives in an environment whose shape resembles the letter T (see Fig. 3.11). The robot's task is to drive from the initial position located at the bottom of the longest corridor, reach the T-junction and then turn to the correct goal (left or right). The correct turning decision at the T-junction depends on the previous input sign received while driving along the corridor (usually a sign at the left/right side of the corridor indicates that the goal is at the left/right arm of the T-maze).

Several systems designed to solve such tasks represent the robot's environment as a discrete world (Bakker, 2002) in order to facilitate the learning of the task. Sometimes the world's representation is not

discrete, but instead some models are designed with event extraction mechanisms which produce abstract signals from the robot sensory cues to the control module (Linaker and Jacobsson, 2001). Recent work has tackled the road sign problem with a continuous world representation (Rylatt and Czarnecki, 2000; Ziemke and Thieme, 2002; Kim, 2004). Most approaches to the road sign problem are based on recurrent neural networks (Ulbricht, 1996; Rylatt and Czarnecki, 2000; Ziemke and Thieme, 2002). The work in Ziemke and Thieme (2002) is based on neuromodulation of synaptic weights in higher-order Recurrent Neural Networks (RNNs) to solve the T-maze task. This means that the sensory-motor mapping (synapses) can be modified while the robot is navigating as a mechanism of short-term memory. This synaptic plasticity is evolved by a standard genetic algorithm. However, for simple T-mazes the resulting controller becomes purely reactive and follows the left wall as soon as the light sign appears at the left side, in contrast to the current work which does not yield wall following behaviors.

In Kim (2004), evolutionary multi-objective optimization is used to evolve finite state controllers in the T-maze task, whose control task is simplified by forcing the robot to first reach the T-junction. In that work, a detailed analysis of the required internal memory for the T-maze task is accomplished.

Reinforcement learning with Long Short-Term Memory (LSTM) is the approach used in Bakker (2002) to solve non-Markovian tasks with long-term dependencies between relevant events such as the T-maze task. A specific gated RNN architecture is used to approximate the value function of a reinforcement learning algorithm. The environment of the agent is discrete, made up of connected squares, and it can execute one out of 4 actions: move North, East, South or West.

This section uses Reservoir Computing as an efficient tool for training robot controllers to solve the T-maze task. The main advantages of this approach are threefold: simplicity of the black-box modeling method; efficient and fast training of RC-based controllers; and integration of reactive and sequential behavior in a single control module. The latter implies that the RC network will learn two different types of competences: reactive and deliberative. The reactive part is given by robot navigation with collision avoidance

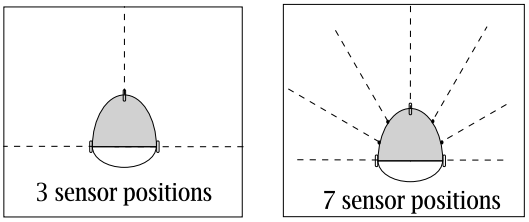


Figure 3.10: Robot models for the T-maze task.

abilities while the deliberative part corresponds to the sequential task of reaching the T-junction and turning to the correct goal depending on the previous cue received. It will be shown that such temporal control tasks are well modeled with RC networks.

3.3.1 Robot Model

The simulation of the road sign problem in the form of a T-maze task is accomplished using the SINAR simulation environment. The SINAR robot model is described in Section 3.2.1 and shown in Fig. 3.1 (or in the Appendix B.1). For the following experiments, obstacles are represented by blue objects whereas the light sign in the T-maze is simulated by a red object. Furthermore, the robot model has either 3 or 7 sensor positions (see Fig. 3.10), and the speed is limited to [0, 17] distance units ($d.u.$).

3.3.2 Experiments

The environments used for the experiments are shown in Fig. 3.11. The task of the robot is to drive from the initial position until the T-junction and then turn left/right if the sign previously appeared at the left/right side of the longest corridor. Environment B has a two times longer corridor than environment A. With that, one can observe how the task is solved when a longer delay between the cue (light sign) and the subsequent response (turning) is required.

The experiments are divided in three stages: **acquisition of the training dataset; training the RC-based robot controller; and testing of the resulting robot controllers**. These three steps are executed for each environment, i.e., a controller trained with data from environment A is only tested in the same environ-

ment. The **first stage** consists of using the robot simulator to generate a training dataset with examples of navigation trajectories in the respective environment. These training samples, containing the robot's sensory inputs and actuators, are recorded by driving the robot through the T-maze using a simple set of rules, e.g., *go from the initial position until the T-junction, then turn slowly to the left arm if the light sign was at the left side previously.* The dashed line in Fig. 3.12(a) represents an example of a trajectory generated by such algorithm. Around 50 examples are collected for the training dataset which considers starting robot positions randomly chosen in the interval [-10,10] *d.u.* for X and Y coordinates as well as initial robot headings randomly generated from the interval [-15,15] degrees.

After data acquisition, the **second stage** consists of training the RC network using the previous collected examples, containing the sensory-motor pairs. The **last stage** corresponds to testing the RC-based controller in the T-maze[5].

The average number of timesteps for the realization of the T-maze task by the algorithm which generates the training dataset is 26.3 timesteps for environment A (standard deviation of 1.6) and 34.9 timesteps for environment B (standard deviation of 1.5). In the testing stage, the T-maze task has to be accomplished in 38 and 46 timesteps for environments A and B, respectively (this was arbitrarily set). These settings do not include the first 20 timesteps in the data acquisition stage as well as in the testing phase, during which the robot stays still, which are used to warm up the reservoir and are discarded in the training of the readout layer due to a transient response of the reservoir (which has initial state $\mathbf{x}(t) = \mathbf{0}$).

At each timestep, Gaussian noise is added to the robot's actuators from the distributions $N(0, 2)$ for the robot turning (in degrees) and from $N(0, 0.5)$ for the robot speed (in *d.u.*). This noise on actuators is considered in the data acquisition stage as well as in the testing stage.

The sensory input in the training observations are 5% noisy, that is, Gaussian noise from $N(0, 0.05)$ is added to distance and color

[5]The testing stage is based on the real-time communication between the Matlab process implementing the RC network and the robot simulator (implemented by TCP/IP sockets).

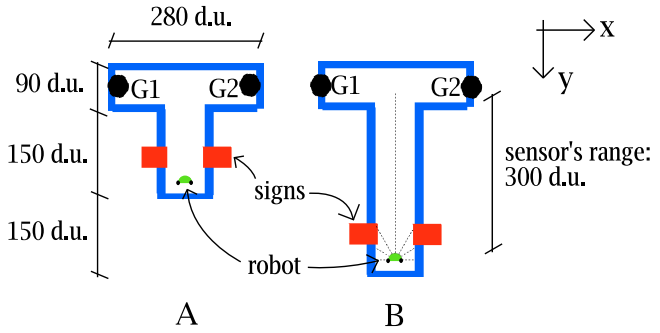


Figure 3.11: Environments used for the experiments. The robot only *sees* 1 sign at a time. A sign at the left (right) indicates that the goal is at the left arm, in G1 (right arm, in G2).

sensors (which means $N(0, 15)$ in distance units), whereas (Rylatt and Czarnecki, 2000) only considers noise-free data for a particular version of the road sign problem solved with Elman networks. The testing stage considers either 1% or 5% Gaussian noise on the robot sensors (this will be stated accordingly in the text).

3.3.3 Settings

The reservoir configuration shown in Table 3.4 is used for all experiments in this section. The inputs to the network are distance and color sensors totaling either 6 inputs (if the robot model has 3 distance sensors and 3 color sensors) or 14 inputs (if the robot model has 7 distance sensors and 7 color sensors). The reservoir is composed of 500 sigmoidal nodes. The readout layer has 2 output units which correspond to the robot actuators of turning (or direction adjustment) and speed (distance traveled per timestep).

3.3.4 Results

In this section, it is investigated how the number of sensors in the robot model and the noise level on the sensory readings affect the performance of the RC-based controller on the T-maze task. This analysis is made for both environments A and B.

An example of the robot's trajectory in the T-maze of environment A is shown in Fig. 3.12(a). The solid line which connects the robot positions at each timestep represents the trajectory of

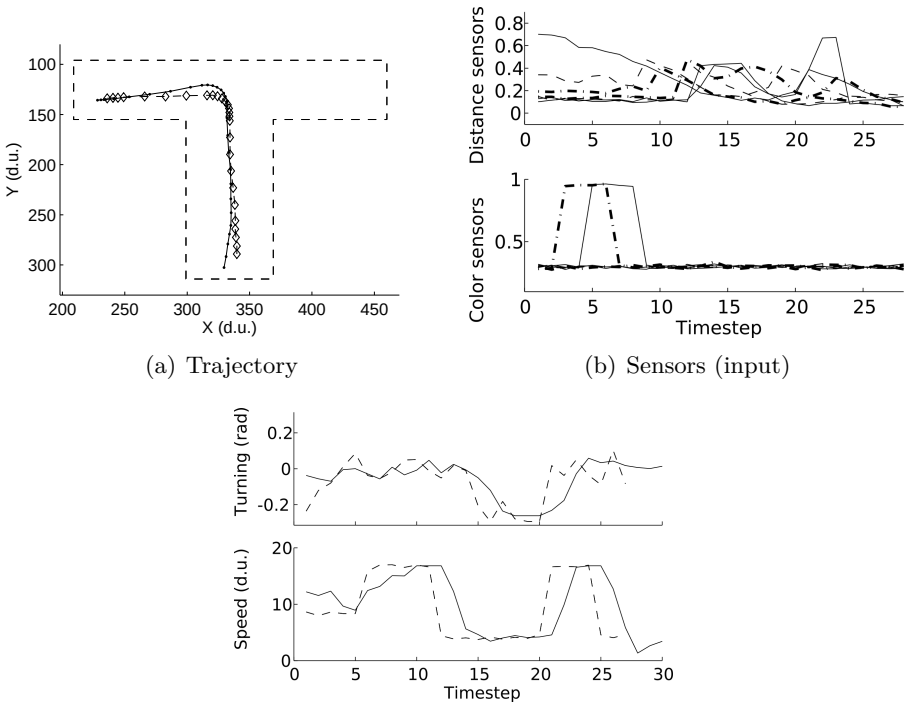
Table 3.4: Parameter configuration for the T-maze task

Number of input channels	$n_i^1 = 6$, $n_i^2 = 14$
Input connection fraction	$c_i^r = 0.2$
Input scaling	$\upsilon_i^r = 0.1$
Input downsampling	$d_t = 1$
Input to output connections	yes
No bias	
Reservoir size	$n_r = 500$
Reservoir connection fraction	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) = 0.9$
Leak rate	$\alpha = 1$
Number of output channels	$n_o = 2$
Output feedback to reservoir	no

the robot driven by the RC network whereas the dashed line which connects small boxes represents an example of a trajectory used for training. The corresponding sensory readings and robot actuators are given in Fig. 3.12(b) and Fig. 3.12(c), respectively.

Fig. 3.12(a) shows that the control task, composed of reactive and sequential behaviors, is smoothly performed by a single control module, i.e., the RC network. After training, the RC network can drive the robot exclusively based on sensor data and can hold the past information for posterior decision making. The recurrent pathways in the reservoir yield a fading memory which is crucial for solving the T-maze task. Traditional feedforward neural networks are not capable of this (Rylatt and Czarnecki, 2000). Furthermore, it takes at least 5 timesteps between the last perception of the sign in the corridor (timestep 9) and the start of the turning movement (timestep 14) in this small T-maze.

The robot trajectory given by the trained RC network in the T-maze of environment B is shown in Fig. 3.13. It can be seen that the time gap between the cue received in the corridor and the decision making at the T-junction can be even greater, in this case 18 timesteps, while the task is still solved correctly. This capacity to hold past stimuli is related to the memory capacity of reservoirs (Jaeger, 2002a), which in turn is influenced, for example, by the size of the reservoir, non-linearity of the units, the input scaling and the



(a) Trajectory

(b) Sensors (input)

(c) Actuators (output)

Figure 3.12: Plots for the robot trajectory (a), sensory readings (b), and actuators (c) considering that the sign appears at the left side of the corridor in the T-maze of environment A. The robot model has 7 distance sensors and 7 color sensors which are 1% noisy. (a) The solid line represents the robot trajectory driven by the RC network in the testing stage (with duration of 38 timesteps), whereas the dashed line corresponds to an example of desired trajectory (with duration of 26 timesteps) included in the training dataset. (b) The corresponding 14 inputs of the RC network (i.e., 7 distance sensors and 7 color sensors readings) in the testing stage for 30 timesteps. (c) The robot's actuators given by the output of the RC network as a solid line (for 30 timesteps) and by the example trajectory as a dashed line (for 26 timesteps).

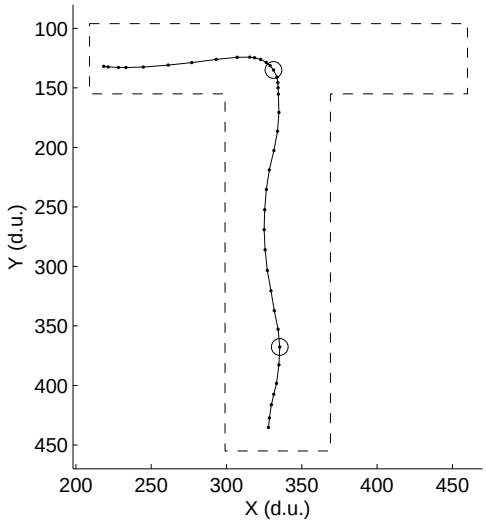


Figure 3.13: Plot for robot trajectory driven by the RC network in environment B. Circles represent the moment in which the robot loses the sight of the sign and the starting time of the turning movement at the T-junction. This time gap is 18 timesteps. The robot model has 7 distance sensors and 7 color sensors which are 1% noisy.

leak rate when reservoir units are leaky integrators.

For evaluating the stability of the preceding results, statistics are generated for environments A and B, considering different noise levels on sensory readings (1% or 5%) and distinct robot models (with 3 or 7 sensors). The mean performance for each combination is shown in Tables 3.5 and 3.6, and is computed after evaluating 30 times each of 10 randomly generated reservoirs in the T-maze for that specific combination. Thus, each cell in the table is the average performance on 300 runs in the T-maze, where the number of runs are split in half for left and right goals. The performance is measured by the percentage of successful trajectories. A run is considered successful if the robot reaches the inner part of the correct arm of the T-maze at the final timestep. For instance, the trajectory shown in Fig. 3.13 is considered successful because the last point of this trajectory has an abscissa which is lower than the abscissa of the left wall of the main corridor ($300 d.u.$).

From the tables, it can be observed that a robot model with

Table 3.5: Percentage of correct trajectories for 1% noise on sensors

	3 sensors		7 sensors	
	Left	Right	Left	Right
Environment A	**58%**	**62%**	**95%**	**93%**
Environment B	**37%**	**33%**	**82%**	**81%**

Table 3.6: Percentage of correct trajectories for 5% noise on sensors

	3 sensors		7 sensors	
	Left	Right	Left	Right
Environment A	**67%**	**70%**	**93%**	**87%**
Environment B	**28%**	**26%**	**69%**	**69%**

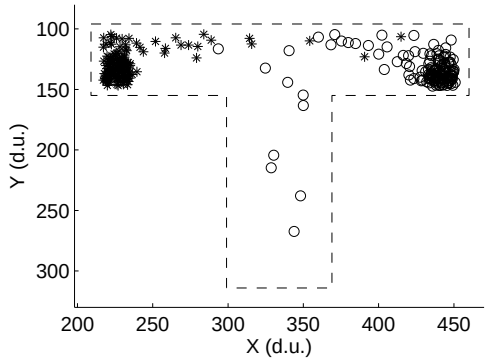


Figure 3.14: Distribution of ending robot positions in environment A. Each asterisk/-circle represents the final robot position after 38 timesteps during which the RC network drives the robot when the goal is located at the left/right arm of the T-maze. There are 300 final robot positions resulting from 10 randomly generated reservoirs which are simulated 30 times each.

7 sensors provides important additional information for solving the T-maze task correctly when compared to a robot model of 3 sensors. With more sensory information, performance increases by 37% for environment A and by 45% for environment B, considering the left goal in Table 3.5. Another observation is that the effect of increasing the noise level on the sensory readings mainly affects the experiments on environment B, specially the ones considering the robot model with 7 sensors. In this case, the degradation in perfor-

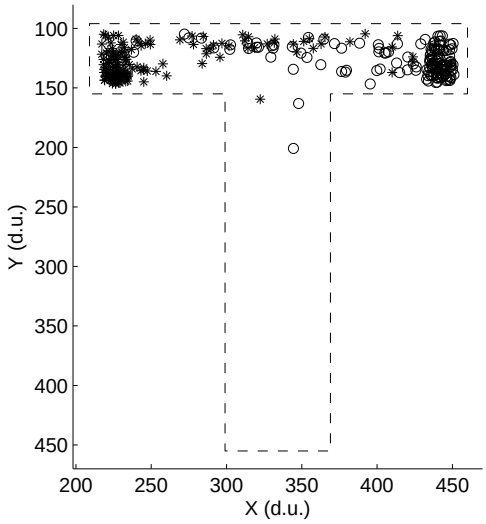


Figure 3.15: Distribution of ending robot positions in environment B. Each asterisk/circle represents the final robot position after 46 timesteps during which the RC network drives the robot when the goal is located at the left/right arm of the T-maze. There are 300 final robot positions resulting from 10 randomly generated reservoirs which are simulated 30 times each.

mance is up to 13%. Therefore, experiments in a bigger T-maze take more advantage of the addition of extra sensory information. Furthermore, higher noise levels negatively influence difficult T-maze tasks, where the difficulty is directly related to the size of the main corridor, i.e., the time gap between cue and required response.

The results considering the robot model with 7 sensors and a noise level of 1% are graphically shown in Fig. 3.14 and Fig. 3.15 for environments A and B, respectively. These figures show the position of the robot at the final timestep of each run[6]. These positions are represented by circles and asterisks which indicate that the sign (cue) for the corresponding run appeared at the right and left sides of the corridor, respectively. There are 300 points for each figure which represent distinct runs in the respective T-maze. As expected, circles are concentrated on the right arm whereas asterisks

[6]Note that only the final position of the robot is evaluated in the figure, despite the robot can still collide against a wall during the intermediate steps (a collision will cause a step back and a small change on the direction of movement.)

are located mainly on the left arm.

3.4 Discussion

There are limitations regarding the architecture used for modeling these robot navigation tasks. As the reservoir has no feedback from output units, the realization of any task is restricted by the size of the reservoir, which is the main factor for memory capacity (Jaeger, 2002a). Bigger reservoirs have more memory to be used in modeling more behaviors and hold temporary past stimuli for longer time spans. Thus, the first measure to be taken to scale these tasks is to increase the number of reservoir units.

Using leaky integrator reservoir units (Section 2.4) is another way to boost the reservoir's memory, by slowing down the dynamics in the reservoir. Using leak rates, each unit maintains its own internal state. A drawback of this approach is that *slow* reservoirs do not react quickly to the input signals, which may be a problem when fast dynamics in the sensory-motor coupling is required, e.g., to avoid collision against obstacles.

3.5 Conclusion

This chapter has presented two types of navigation tasks which are modeled with a single RC network. A set of different sensory-motor couplings (or behaviors) is modeled through a supervised learning process which generates examples of desired behaviors or trajectories. Each of these behaviors get embedded in a sub-space attractor in the dynamical system space after training the RC network.

Modeling multiple reactive behaviors (i.e., navigation attractors) or even delayed control tasks (in the case of the T-maze task) has been achieved by the use of salient external switches coming from the environment or modeled explicitly by an extra input channel. These external switches enables to embed each behavior into a *sub-space attractor* in the dynamical system space. These sub-space attractors are made possible by three reasons:

reactive behaviors the reactive nature of navigation behaviors

yields robust attractor-like sensory-motor coupling throughout the environment space;

shifting into sub-spaces training RC-based controllers is accomplished by linearly discriminating a set of sensory-motor couplings, each type (or behavior) shifted into a sub-space of the dynamical system by the use of sensory inputs;

coupling of controller and environment during testing, a tight coupling of RC-based controller and environment makes it possible to achieve attractor-like trajectories in the environment space.

Now, in Chapter 4, RC networks will be used to learn implicit spatial representations of an environment, making it possible to predict the location of a robot as it navigates, based solely on noisy information of distance sensors from small (simulated and real) mobile robots. With this, a new internal mechanism for switching between behaviors is made possible. These *context switches* are fired whenever the robot leaves one room and enters another one (i.e., by crossing doors connecting rooms), once the predictive outputs modeling the location also change their values.

Thus, this chapter serves as a proof of concept for learning multiple reactive behaviors using external switches, and as a base for enabling more complex sensory-motor coupling based on implicit learned context switches (focus of Chapter 5).

4
Robot Localization

In order to implement complex goal-directed behaviors for autonomous navigation systems in small mobile robots under a bottom-up approach, it is necessary to extract useful environmental features using only few noisy sensory inputs such as inexpensive infra-red distance sensors. In this chapter, Reservoir Computing networks are used to model spatial environment features by predicting the location of simulated and real mobile robots based on local information from few noisy distance sensors, overcoming sensor aliasing problems which are disambiguated due to the fading memory property of the dynamical reservoir.

4.1 Introduction

Traditional algorithms based on the Simultaneous Localization and Mapping (SLAM) concept are, in many cases, expensive to implement due to high computational and memory demands and also hold uncertainties during the calculation of the robot's pose (Bailey and Durrant-Whyte, 2006) (see Section 1.2.2). They usually need high precision ranging data from, for example, a 2D laser range scanner (or a camera in the case of Visual SLAM). Laser scanners are currently still expensive, consume a considerable amount of power, and cannot easily be applied to small mobile robots. Low-cost, small and lightweight robots that have a high battery autonomy will thus not be able to use a SLAM based approach in most cases. These robot platforms usually only have access to a limited number of ranging

sensors which are low range and have high noise[1].

Instead of trying to hard-code a priori knowledge and explicitly modeling the environment of a mobile robot under the *top-down* approach, this chapter is based on designing intelligent systems which learn internal models solely through local sensory information, thus following a *bottom-up* approach. One of the advantages of using these systems is that they are inherently robust to noisy sensors and unpredictable events in dynamic environments.

Reservoir Computing networks are used in this chapter to detect complex events as well as to predict the position of mobile robots in their environments. For this, RC networks form an implicit spatial representation which disambiguates the sensory space as the low-dimensional input from distance sensors are projected into the high-dimensional dynamic space of the reservoir (acting as a temporal non-linear kernel). The short-term fading memory of RC networks is crucial for solving the aforementioned tasks. It is not only the instantaneous sensory inputs that are needed to solve these tasks, but also the sensory history (Schönherr et al., 2001) and dynamics.

It has already been shown in (Hertzberg et al., 2002) that RC can be used to detect events in an autonomous robot setting. This chapter extends these results by also considering dynamic environments for event detection, and goes largely beyond that work by using it to construct implicit maps of complex, symmetric environments for robot localization.

The idea of employing a neural network as a localization model for the robot is also inspired by biological systems. Experiments accomplished on rats show that their hippocampus forms activation patterns that are associated with locations visited by the rat. These so called *place cells* encode the spatial location of the animal into its environment. They fire when the animal is in a particular location (O'Keefe and Dostrovsky, 1971) (see Section 1.2.4.1 for more details).

The experiments in this chapter are performed using not only two different robot simulators, but also real robots in unstructured

[1]It is relevant to note that this thesis does not aim at directly competing with or substituting the SLAM-based algorithms, although it may seem so. Instead, it focus on RC-based approaches to robot navigation and environment learning in a biologically inspired black-box fashion.

environments. The datasets generated by simulators or obtained from the real robot are used to train an RC network in a supervised learning way to detect events as well as to predict the robot location. Both tasks are based solely on short-range, high-noise sensory information, typically found in small and inexpensive mobile robots. While in this chapter the training is done in a supervised fashion, Chapter 7 tackles the unsupervised learning of locations, in a way probably closer to the way place cells in biological systems are formed.

After designing an RC-based location detector, new possibilities open for goal-directed navigation using small mobile robots which could be applied to domestic tasks such as floor cleaning. Additionally, generative extensions of the RC network proposed here will be introduced in Chapter 6, which can be used in tasks such as path planning and predictive modeling of behaviors.

4.1.1 Types of spatiotemporal detection tasks

In a first part, it will be shown that an RC network can be used to perform detection of arbitrarily predefined spatio-temporal events during robot navigation tasks in dynamic environments. A description is given next.

events are given by well defined spatiotemporal occurrences in the environment of a mobile robot, i.e., they occur in a certain place during a particular interval of time and can also be related to which orientation the robot approached that certain place. See Fig. 4.1 for an example.

Moreover, the same type of RC network can be used in different types of localization tasks with specific levels of abstraction. The three levels of spatial abstraction used in this chapter are given below:

robot's pose is defined by the robot coordinates and heading in the global coordinates frame. It is the lowest level of abstraction. The dashed trajectory with arrows in Fig. 4.1 shows the robot's pose over time in space. This is referred as the *pose dimension*.

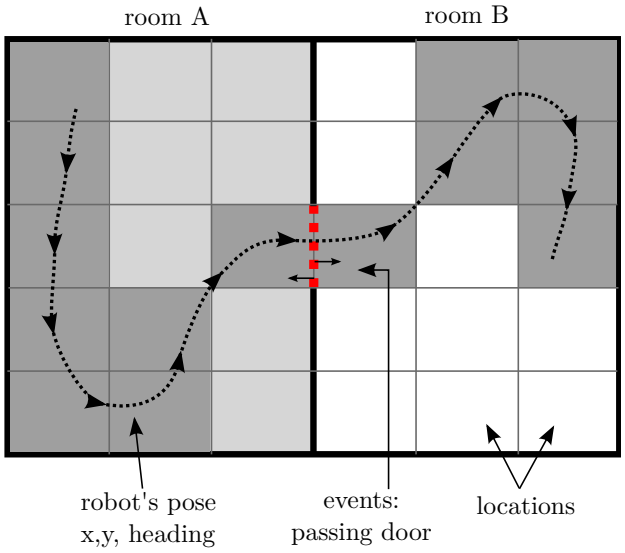


Figure 4.1: Hypothetical environment with 2 rooms for showing the detection of spatiotemporal features of an environment: event detection and the different levels of abstraction (or dimensions) in robot localization tasks. The two events *passing through a door* coming from different robot orientations are marked by two small arrows near a red colored dashed line which represents the firing of both events. The black dashed line and arrows represent the *robot's pose* over time, which forms a fine trajectory in space (*pose dimension*). More abstract concepts of localization is given by *locations* which are small delimited sub-regions of similar size (such as a square) in an environment. The trajectory of the robot in the *location dimension* is defined by sequence of squares colored in dark grey. The most abstract spatial concept is given by whole environmental *rooms*, e.g., forming bigger areas separated by doors. Room A is colored in light grey, whereas room B is in white.

location is given by a small delimited sub-region of the environment. It is a more abstract concept than robot's pose. At the *location dimension*, the environment is divided into small discrete areas such as squares in Fig. 4.1.

room is formed by a large environmental sub-region, usually bounded by walls and doors. It is the most abstract spatial concept used in this chapter. The *room dimension* is formed by these several room sub-regions as shown in Fig. 4.1.

4.2 Robot Models

Two simulated robot models and a real robot are used in the following experiments. The SINAR robot model, simulator and controller are described in Section 3.2.1 from the previous chapter. The second robot model is the e-puck robot, which is used in simulated and real experiments, as described in the next section.

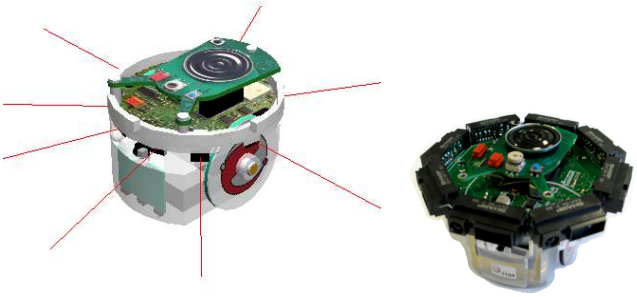
4.2.1 E-puck Robot

For convenience, the following information on the e-puck is also provided in the Appendix B.2.

4.2.1.1 Description

The e-puck (Mondada, 2007) is a small differential wheeled robot which was built primarily for education purposes, but has been largely adopted in research as well. The mobile robot has a diameter of 7 cm and is equipped with 8 infra-red sensors which measure ambient light and proximity of obstacles in a range of $(0-5]$ cm originally, which effectively restricts the ability to read distances to obstacles. Because of this, an extension turret for the real e-puck robot has been built with 8 longer-range infra-red sensors capable of measuring distances in the interval $[4-30]$ cm (see Fig. 4.2(b)). The actuators of the robot are 2 stepper motors, where the maximum possible speed is 1000 steps per second.

The experiments in this thesis are accomplished with a simulated version of the e-puck robot as well as with the real e-puck robot.



(a) Webots e-puck (b) **Extended Real** e-puck

Figure 4.2: E-puck robot models used in this chapter. (a) E-puck robot from Webots simulation environment. (b) Real e-puck robot extended by an additional turret containing 8 infra-red sensors capable of reading distances from 4 cm to 30 cm.

The Webots software (Michel, 2004), used for simulations, provides a physics model of the e-puck robot (Fig. 4.2(a)), i.e. the simulator detects collisions and simulates physical properties of objects, such as the mass, the velocity, the inertia, the friction, the spring and damping constants, etc. A simulated timestep in Webots takes 32 ms.

The simulated e-puck model is used with two types of sensors regarding their range: $[0-5]$ cm for event detection experiments and $[0-15]$ cm for robot localization experiments, in order to provide sufficiently rich data for learning the respective tasks. The velocity of the simulated robot is limited to $[0.6,\ 3]$ cm/s.

Table 4.1: Robot models used in this chapter

	SINAR	sim.e-puck variant 1	sim.e-puck variant 2	real e-puck
No. Dist. Sensors	17	8	8	8
Range of Dist. Sens.	$[0, 300]$ d.u.	$(0, 5]$ cm	$(0, 15]$ cm	$[4, 30]$ cm
Noise on sensors	$N(0, 60)$ d.u.	$N(0, 1.5)$ cm	$N(0, 4.5)$ cm	inherent noise
Speed	0.28 d.u.	$[0.6, 3]$ cm/s	$[0.6, 3]$ cm/s	$[0.198, 5.08]$ cm/s
Physics model	no	yes	yes	–

4.2.1.2 Robot Controller for Webots

The controller for the simulated e-puck robot, used to generate training data, is made of a simple algorithm which follows points from a predefined trajectory in the environment. Its speed can be 3 cm/s, 1.25 cm/s or 0.63 cm/s.

4.2.1.3 Robot Controller for Real Environments

For recording datasets containing the robot's sensor readings, a controller written in Matlab steers the e-puck robot through a Bluetooth connection. This controller performs basic wall following throughout the environment and it switches randomly to left or right wall following with a certain probability τ[2]. When the robot switches from right to left wall (or vice-versa), it may generate ellipsoidal trajectories inside a room until it finds a wall to follow. Thus, the robot may stay navigating inside a particular room for a random time interval. The results shown in this chapter considers that $\tau = 0.03$, which practically means that there is a probability of approximately 60% for inverting the direction of movement while the robot is navigating inside one of the rooms.

One iteration, for reading the distance sensors as well as for motor actuation, lasts 200 ms. The speed actuator is limited to the interval $\pm[15, 385]$ steps/s (or $\pm[0.198, 5.08]$ cm/s).

The eight distance sensors are sequentially read in groups of 2 while the robot is moving, that is, there are 4 cycles of sensor reading, where each cycle corresponds to 2 simultaneous readings. Considering an acquisition time of 25 ms on average for a cycle, the total time spent on sensor reading is between 100 and 120 ms. Any resulting inconsistencies from this sequential sensor reading during robot movement are not corrected, so that learning has to cope with this additional problem.

The input signal $\mathbf{u}(t)$ for the reservoir state update equation is built by recording the eight distance sensors during robot navigation and scaling them to the interval $[0, 1]$. For analysis purposes, the robot position and orientation are estimated using pictures taken from a fish-eye camera placed on a structure localized above the

[2] τ is the probability of changing the movement direction at each second and determines the randomness of the robot movement.

environment. Robot recognition and pose tracking are accomplished using the ReactiVision software (Kaltenbrunner and Bencina, 2007).

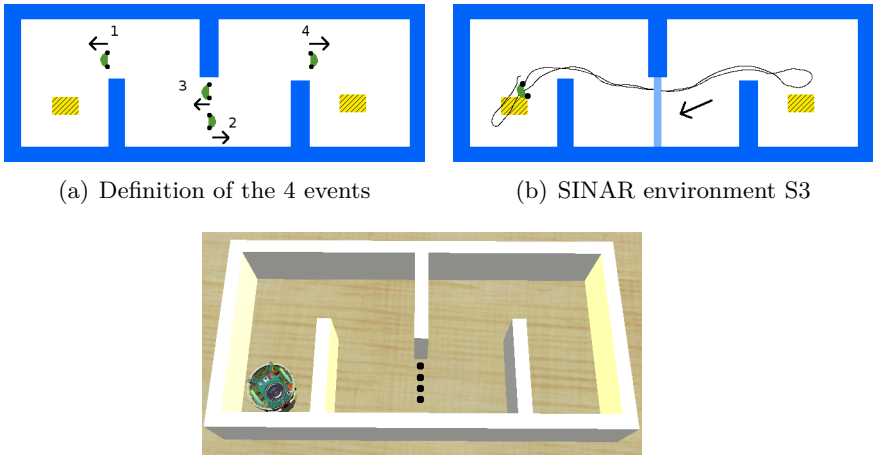
4.3 Event Detection for Mobile Robots

Event detection in noisy environments is not a trivial task. There can be very similar scenes from the robot's perspective so that precise event detection becomes very difficult to accomplish (Jaeger, 2002a). The goal here is to achieve efficient event detection using reservoir computing. The detection of events from raw sensory data is much related to the so called symbol grounding problem (or anchoring) in robotics (Harnad, 2003). Several applications are appealing in this context once deliberative robotic systems can benefit in several ways from efficient meaning extraction from sensory data (Harnad, 2001; Vogt, 2001; Rosenstein and Cohen, 1999).

Examples of event detection in mobile robot navigation include the detection of new objects in an environment, recognition of momentaneous situations such as: passing through door A, entering room B, cyclic robot trajectories, etc. The task could also be defined in a more complex way such as the detection of a sequence of events (e.g., enter and exits room B through the same door).

4.3.1 Environments

Two experiments are conducted for the event detection task. The environments used for SINAR and e-puck are shown in Fig. 4.3(b) and Fig. 4.3(c), respectively. The first environment is composed of a large (blue) corridor with a (yellow) target at each end (they appear as dark and light gray objects in black and white format). During simulation, the robot keeps navigating through the corridor and capturing the targets that are sequentially put back in the same location. A *blinking* object located in the middle of the corridor, that disappears and re-appears by a random time interval, can force the robot to change direction by blocking its way. In the second environment, the e-puck robot follows a predefined trajectory which continually visits the entire environment. Its trajectory can be inverted when it reaches the middle of the environment marked by a dotted line in Fig. 4.3(c) with a probability of 50%.



(a) Definition of the 4 events

(b) SINAR environment S3

(c) e-puck environment E1

Figure 4.3: Environments used for the event detection task. (a) Four events are labeled and shown graphically (by arrows). (b) SINAR environment with a *blinking* obstacle in the middle of the corridor, indicated by an arrow. A typical robot trajectory (after controller learning) can be seen in the figure. Two boxes in the environment are used as targets for the robot. (c) e-puck simulation environment (the 4 events are defined similarly). The dotted line represents a decision point which makes the robot cross the line or go back with equal probabilities.

There are four possible events of predefined duration and location, which are labeled in Fig. 4.3(a). The interpretation should be: when the robot passes through a predefined location with a specific heading, an event should be detected for a short-time interval, e.g., entering the corridor corner area, passing through the middle of the corridor.

4.3.2 Settings

Experiment 1 is accomplished considering the SINAR environment S3 and experiment 2 takes place in the e-puck environment E1. Experiments 1 and 2 have 126.000 and 120.000 timesteps of simulation time, respectively. Parameter configuration for both experiments are shown in Table 4.2.

In order to match the timescale of the sensory input to the timescale of the event detection task, both data downsampling (d_t) and leak rate (α) in the reservoir are used. Although resampling

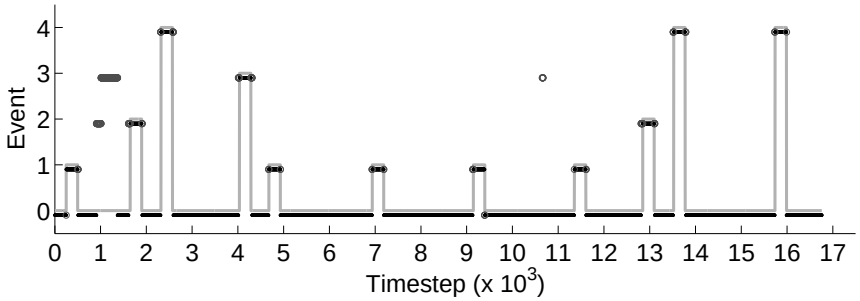
Table 4.2: Parameter configuration for event detection

Model	SINAR	e-puck
Number of input channels	$n_i = 18$	$n_i = 10$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.3$
Input scaling	$\upsilon_i^r = 0.2$	$\upsilon_i^r = 0.2$
Input downsampling	$d_t = 20$	$d_t = 15$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 800$	$n_r = 800$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 0.6$	$\alpha = 0.8$
Number of output channels	$n_o = 4$	$n_o = 4$
Output feedback to reservoir	no	no

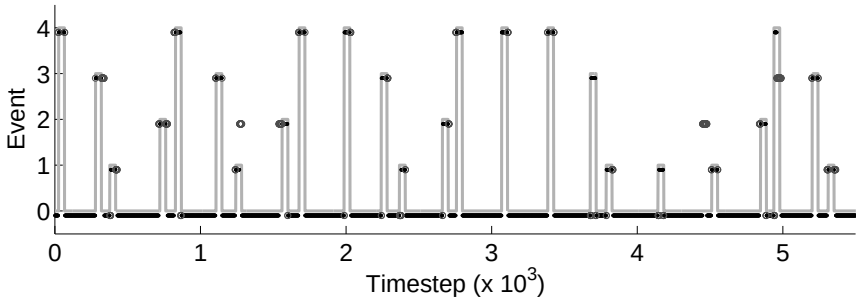
and leak rate are considered equivalent (Schrauwen et al., 2007), it seems that the combination of both methods yields superior performance, as it will be shown in the results, by more finely adjusting the temporal dynamics of the reservoir to the dynamics of the modeling task. The value of the parameters d_t and α was optimized by performing a grid search, i.e., the combination of parameters with highest test performance was chosen.

The inputs to the network for the SINAR model are 17 distance sensors and 1 robot actuator (direction adjustment) while for the e-puck model the inputs are composed of 8 sensors and 2 motor actuators. Although a large reservoir of 800 units is used, smaller reservoirs containing 200 or 400 nodes have shown to also perform very well on average. The readout layer has 4 output units (one for each event detector) which are post-processed by a function g defined in (2.22) in Section 2.7, where the Fisher relabeling for unbalanced datasets (2.24), which is applied to the current event detection task, is also described.

The performance measure considers the number of correctly predicted observations (defined in (2.26)) and uses a 7-fold (8-fold) cross-validation method for the SINAR (e-puck) model. It is important to note that if the dataset is resampled, then the output of the network is upsampled to the original sampling rate of the dataset



(a) Event detection using SINAR model



(b) Event detection using simulated e-puck variant 1

Figure 4.4: Event detection performed by RC network. The gray solid line represents the actual event whereas the black points corresponds to the predicted events. Mispredictions are marked with circles. (a) Using SINAR simulation model (performance of 95.4 % on this test data) (b) Event detection using e-puck simulation model (performance of 93.1 % on this test data).

so that the performance is correctly calculated.

4.3.3 Experimental Results

The results are shown in Fig. 4.4. For both robot models, the RC network performs very well by achieving 95.4% and 93.1% of classification performance on test data. Although the events during a robot run are not periodic, the 4 classes of events are correctly detected during the whole simulation, except for a few mispredictions. Most of the errors are explained by the temporal resolution of the RC-based detector, that is, the reservoir is sometimes not accurate enough in the beginning/end of an event (i.e., in the temporal boundary of events).

In Fig. 4.5, it can be seen how the downsampling rate of the

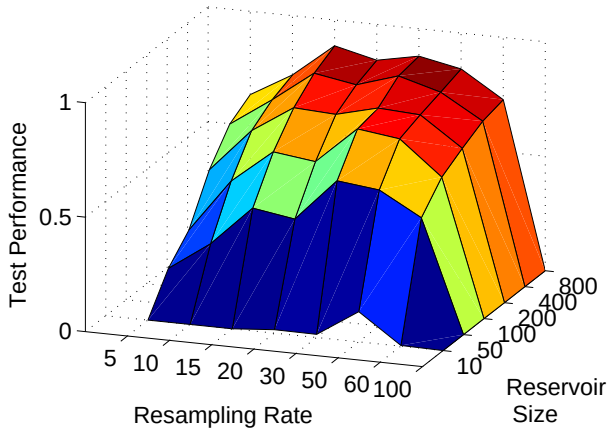


Figure 4.5: Test performance for resampling rate vs. reservoir size. Each point of the plot is the mean performance (correct classification rate) over 30 runs for the event detection task using the e-puck robot model.

dataset (d_t) and the size of the reservoir influence the test performance on the event detection task using the e-puck robot model. If a dataset is downsampled by $d_t = 10$, for instance, the resulting dataset will be 1/10 smaller than the original one, and will be smoothened similarly to a low-pass filtering of the input. Fig. 4.5 shows that a downsampling of 15 timesteps is the optimal choice. It is possible to observe that when d_t is bigger than 30 timesteps, as the downsampling rate increases the performance deteriorates. The figure also shows that bigger reservoirs (with more neurons) have more memory (while require less resampling), thus increasing performance.

Statistics on experiments 1 and 2 are given in Table 4.3. Each experiment is evaluated 30 times with different stochastically generated reservoirs and the results are averaged over these 30 runs. The table shows that the results are consistent, with 93.2% and 92.3% of performance on test data for SINAR and e-puck models,

Table 4.3: Performance results for event detection

Model	Timesteps	Train Perf.	Test Perf.	Perf. Events 1, 2, 3 and 4 resp.			
SINAR	126 K	94.7 %	93.2 %	**95.5%**	**95.6%**	**100%**	**99.6%**
e-puck	120 K	93.2 %	92.3 %	86.9%	**97.2%**	**96.7%**	82.6%

respectively.

Despite events 2 and 3 can be considered symmetric, they show distinct performances (respectively, 95.6% and 100%). The reason for different performances may be caused by two factors: one is that the reactive controller which steers the robot may generate different trajectories and thereby distinct streams of sensor activations right before events 2 and 3; another factor is that event 2 may have appeared in the test sequence more times than event 1, thus increasing the chance that event 2 is mispredicted. One can also note that the performance for the e-puck model for events 1 and 4 are worse than the performance for events 2 and 3. This is probably due to the robot controller which exhibits different velocities during navigation (low speed close to events 1 and 4 and high speed near events 2 and 3).

4.4 Robot Localization

The previous section has shown that an RC network can be used to detect complex events in robot navigation with good performance. Now this section tackles the detection of more complex spatiotemporal environment features during robot navigation. Instead of only detecting events, the goal is to predict the current **location of the robot** based on the same kind of instantaneous sensory information, giving rise to a more difficult and interesting problem. This section shows how a reservoir can be used for robot localization. Similar preliminary work which uses a Long-Short Term Memory RNN for this task is described in Forster et al. (2007).

The proposed localization system is built upon the learning of an implicit map of the environment by a RC network. The output layer of the RC network (see Fig. 1.4), represented by the column vector $\mathbf{y}(t)$ in (2.2), creates a spatial representation of the environment solely from few distance sensors. The vector $\mathbf{y}(t)$ results from a linear combination of a temporally processed input vector of distance measurements. In this work, the system receives allothetic signals as input (i.e., external sensory input given by distances sensors) as well as idiothetic signals in the form of actuator feedback. However, **the proposed system acquires the same localization**

properties and presents similar performances for the following experiments whether this actuator feedback is used or not. Neither odometry information nor dead reckoning (*path integration*) are used in this work. Odometry is not biologically plausible: counting wheel revolutions for determining the next position is very unlikely to happen in biological systems. Instead, these systems could have very basic forms of *noisy dead reckoning* with very limited capacity for path integration over extended trajectories. In the case of an omni-wheeled robot, for instance, dead reckoning would hardly work because of intrinsic noise present in movements which are typically based on wheel slippage.

Next, the three levels of abstraction in location prediction, introduced in Section 4.1.1, are formulated as regression or classification tasks.

4.4.1 Pose estimation with output feedback

Robot's pose estimation can be formulated in the form of a regression task where the exact robot coordinates and heading in the global coordinates frame are expected as output. The RC architecture used for this task is shown in Fig. 4.6(a) where the estimated robot's pose at the output layer is fed back into the reservoir. This additional stimulation endows the reservoir with long-term memory (Maass et al., 2006) for the pose estimation task, effectively improving its performance. The fading memory of a reservoir which has no feedback connections from the readout layer is not enough for satisfactory performance. During training, the desired x, y, θ coordinates are *teacher-forced* through the output units into the reservoir (see Section 2.5) while after training the predicted pose is fed back. The continuous valued predictions located at the output layer are externally driven solely from few noisy distance sensor's measurements which are projected into the dynamic non-linear reservoir space.

4.4.2 Location and Room detection

The localization task can be formulated as the detection of subregions of an environment. This can be accomplished by discretizing the environment into a set of delimited areas of specific scales, such as *locations* and *rooms*. A *location* in this work consists of a small

sub-region of size ranging from 1 to 5 times the size of the robot, whereas a *room* is comprised by a much larger sub-region of the environment, usually divided by doors and corridors.

The task is to detect the environmental sub-region in which the robot is currently located. This can be accomplished by using an RC architecture shown in Fig. 4.6(b), where the readout output layer is trained to perform classification (Section 2.7). Thus, each output in this layer corresponds to one sub-region in the environment and should be activated as soon as the robot enters the respective location or room. The example given in Fig. 4.1 shows squared locations in grey which represent the activated outputs of the readout layer as the robot navigates in the environment.

4.4.3 Environments

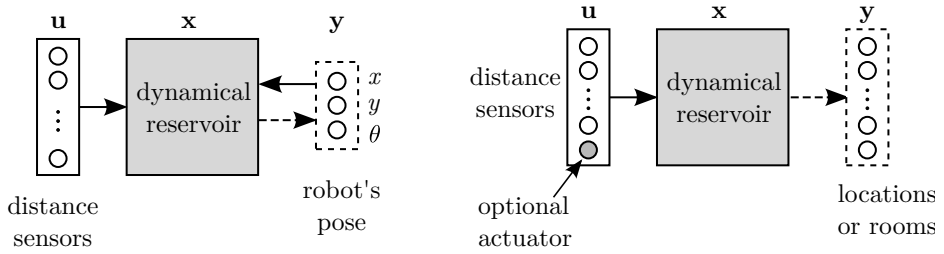
Three different robot models are used to validate the capability of RC networks for modeling the localization of robots in several unknown environments. Whereas SINAR is a fast and flexible simulator for mobile robots, the e-puck model in Webots provides physically realistic 3D simulations. The third model is the real e-puck robot.

4.4.3.1 SINAR robot

The environments used for the **pose estimation task** using the SINAR model are shown in Fig. 4.7. The first environment (S4) is a long T-maze which is used to test the memory capacity of the RC network. When the robot is driving along the longest corridor, the frontal distance sensors do not detect the ending corner of the corridor since they saturate. Therefore, for correct pose detection, it is strictly necessary the use of the short-term memory of the reservoir.

The generation of trajectories is based on (Antonelo et al., 2006)[3] and is explained next. The robot trajectory in environment S5 depends on the current visible target - only a randomly chosen attractive object is visible at a time. The controller keeps on capturing targets indefinitely during simulation, i.e., a target is hidden when

[3]Other controllers could be used for data generation. So, the use of the specific controller is not critical for the results in this chapter. For instance, Braitenberg vehicles (Braitenberg, 1984) could be used instead.



(a) Pose estimation with output feedback

(b) Location or room detection

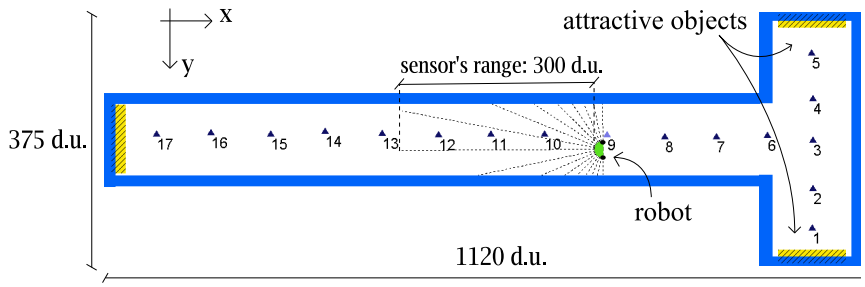
Figure 4.6: Two RC architectures for modeling robot localization. (a) RC architecture for modeling regression tasks with output feedback such as the prediction of the robot coordinates and heading in the global coordinates frame. (b) RC architecture for modeling dynamic classification tasks such as detection of locations and rooms of an environment.

captured and another one is made visible. In environment S5, the robot's trajectory depends on the dynamics of the *blinking* obstacles (see Fig. 4.7) and on Gaussian noise which is added to the motor output. This diversification of trajectories is accomplished in order to make the prediction problem more difficult.

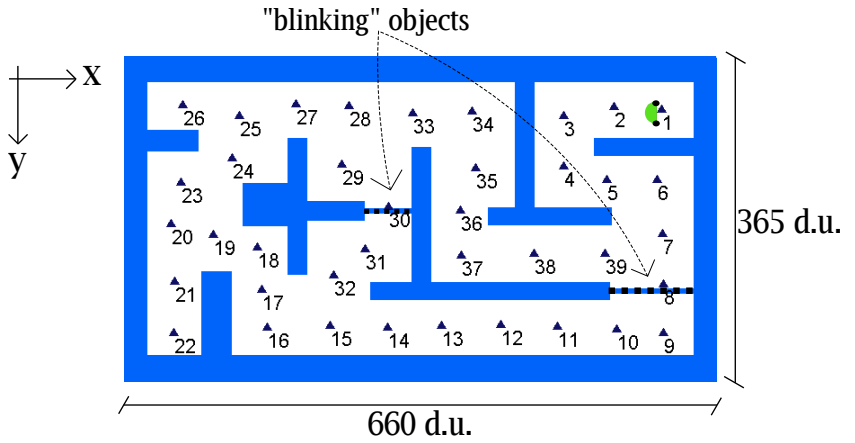
Two maze-like environments S6 and S7 shown in Fig. 4.8 are used for the **location detection task**. Environment S6 contains 64 predefined locations, whose centers are displayed by small triangles labeled by numbers. The second environment S7 has 29 locations distributed in a symmetric map.

4.4.3.2 E-puck robot

The environment E2 shown in Fig 4.9(a) is used for **location and room detection**. It is composed of 4 big rooms with doors connecting them. Fig 4.9(b) shows 30 points distributed in the map which are connected by lines representing possible predefined robot paths between them. When the robot reaches a point which can lead to 2 other possible points, the robot controller decides to choose one of the points with equal probabilities. In this way, the robot may stay in room 3 for a variable time interval in the same run, for instance. The location detection task consists of predicting which one of the 30 points the robot is most close to. The room detection task is based on the map shown in Fig 4.9(c) and consists of predicting the current robot room out of the existing 4 rooms.



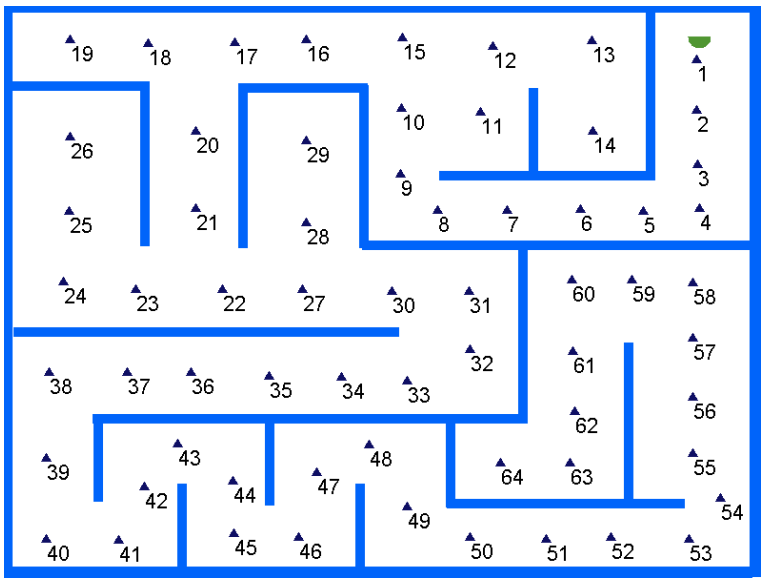
(a) SINAR Environment S4



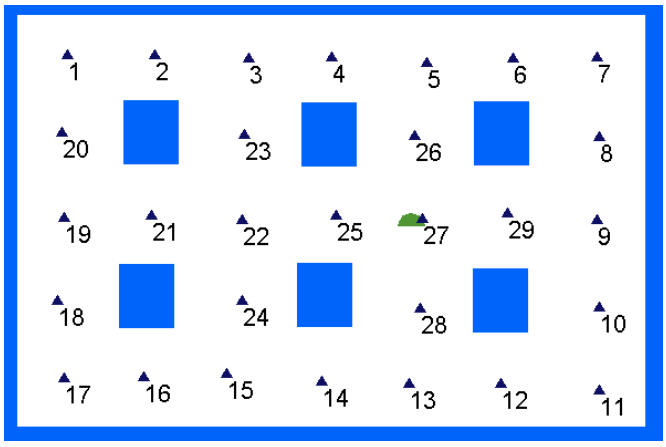
(b) SINAR Environment S5

Figure 4.7: Environments used for the experiments on pose estimation using the SINAR model from Table 4.1. Dimensions are given in *distance units* (d.u.). (a) Long T-maze environment. Locations are marked by small labeled triangles and attractive objects are located at each arm's corner. (b) Complex room environment with two dynamic objects at locations 8 and 30 which *blinks* by random time intervals (blocking or freeing the robot's way).

The environment E3 for the real e-puck, shown in Fig. 4.10, has 3 rooms and a corridor connecting them. A camera mounted above the environment records the position of the robot at each timestep. In this way, the labels for the supervised learning of **locations** and **rooms** can be collected. For generating datasets with recorded sensor readings, the robot controller described in Section 4.2.1.3 is used. The total number of recorded samples amounts to 192.000 timesteps, which corresponds to approximately 11 hours of robot navigation.

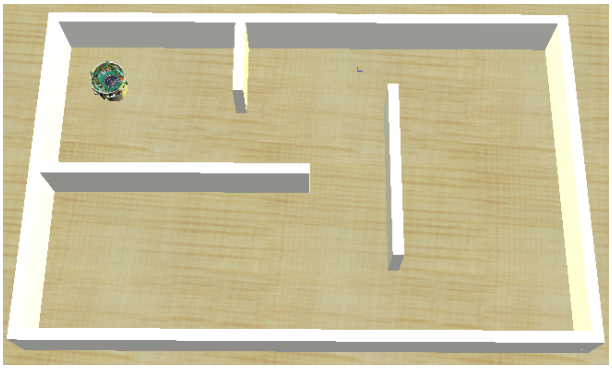


(a) SINAR Environment S6

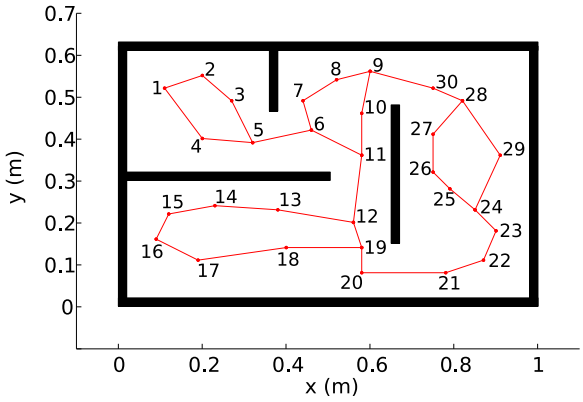


(b) SINAR Environment S7

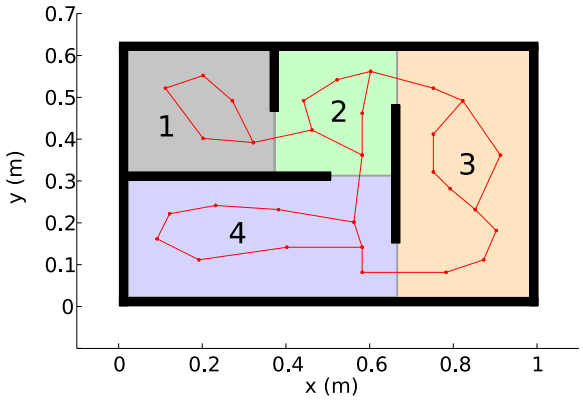
Figure 4.8: Environments used for the experiments on location detection using the SINAR model from Table 4.1. Dimensions are given in *distance units* (d.u.). (a) The first environment is tagged with 64 labels displayed by small triangles. (b) The second environment has 29 labels distributed through very similar areas.



(a) Environment for e-puck (E2)

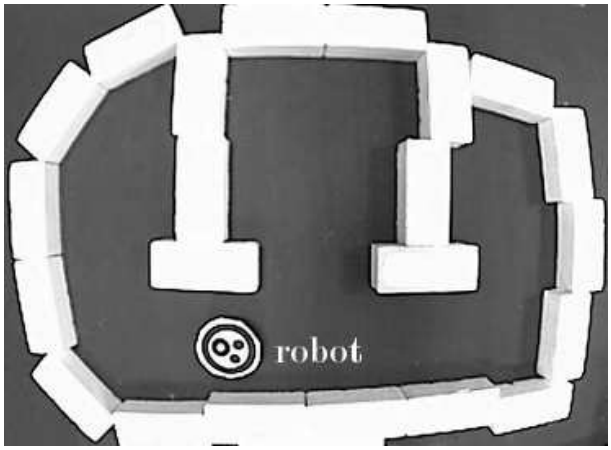


(b) Map for Location detection

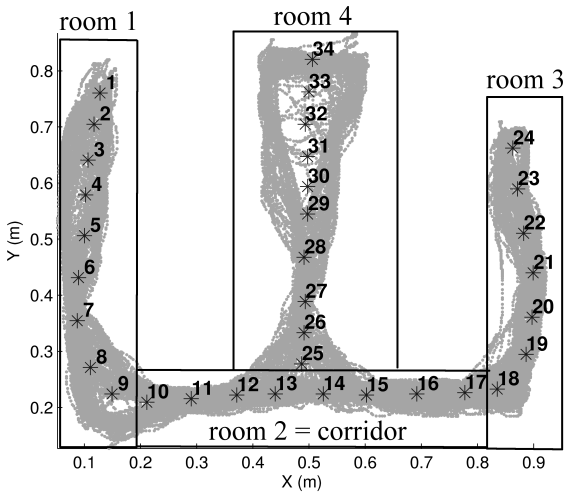


(c) Map for Room detection

Figure 4.9: Environment used for localization experiments using the simulated e-puck variant 2 robot from Table 4.1. (a) the e-puck in its 3D environment. (b) the map of the environment with 30 locations to be detected. Each location is represented by a point and labeled with numbers. The points are connected by lines which represent possible paths between locations. (c) the map of the environment showing the 4 rooms to be detected and the same robot exploring trajectory.



(a) Real environment for e-puck (E3)



(b) Robot trajectory

Figure 4.10: Environment used for localization experiments using the real e-puck robot from Table 4.1. (a) Environment (120 cm × 90 cm) composed of three rooms and one corridor. The position of the robot is tracked with a camera mounted above the environment for analysis purposes. (b) Trajectory in gray generated by the robot controller in environment E3 for 60.000 timesteps (or 3.3 hours), with labeled asterisks representing delimited locations.

Table 4.4: Parameter configuration for pose estimation

Model (Env.)	SINAR (S4)	SINAR (S5)
Number of input channels	$n_i = 17$	$n_i = 17$
Input connection fraction	$c_i^r = 0.4$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 0.4$	$\upsilon_i^r = 0.6$
Input downsampling	$d_t = 30$	$d_t = 50$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 800$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 1$	$\alpha = 1$
Number of output channels	$n_o = 3$	$n_o = 3$
Output connection fraction	$c_o^r = 0.4$	$c_o^r = 0.3$
Output feedback scaling	$\upsilon_o^r = 0.4$	$\upsilon_o^r = 0.025$

4.4.4 Settings

4.4.4.1 Pose Estimation

The parameter configurations for pose estimation on environments S4 and S5 are given in Table 4.4. The inputs to the network are only the 17 distance sensors of the robot. The readout output layer has 3 output units which corresponds to the normalised robot coordinates and heading. Both ridge regression for the output training and the adding of Gaussian noise with variance of 0.001 to the state update equation is used for regularisation. Only using ridge regression is not enough to get stable output feedback behaviour. The downsampled dataset for environment S4 (environment S5) is divided in two parts of 2400 and 1200 samples (5600 and 800 samples) which are used for training and testing, respectively.

4.4.4.2 Location and Room Detection

The configurations used for the experiments on **location** and **room** detection with SINAR, simulated e-puck variant 2, and real e-puck robot models from Table 4.1 are presented in Table 4.5 and Table 4.6, respectively.

The inputs to the network n_i are 17 distance sensors and 1 robot

Table 4.5: Parameter configuration for location detection

Model	SINAR	simul.e-puck variant 2	real e-puck
Number of input channels	$n_i = 18$	$n_i = 10$	$n_i = 8$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.3$	$c_i^r = 0.3$
Input scaling	$\upsilon_i^r = 0.1$	$\upsilon_i^r = 0.9$	$\upsilon_i^r = 2$
Input downsampling	$d_t = 50$	$d_t = 10$	$d_t = 1$
Input to output connections	yes	yes	yes
No bias			
Reservoir size	$n_r = 800$	$n_r = 800$	$n_r = 1200$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.99$
Leak rate 1	$\alpha_1 = 0.6$	$\alpha_1 = 0.05$	$\alpha_1 = 0.005$
Leak rate 2	$--$	$\alpha_2 = 0.8$	$\alpha_2 = 0.01$
Leak rate 3	$--$	$\alpha_3 = 1$	$\alpha_3 = 0.2$
Number of output channels	$n_o = \{64, 29\}$	$n_o = 30$	$n_o = 34$
Output feedback to reservoir	no	no	no

actuator (direction adjustment) for the *SINAR model*; 8 distance sensors and 2 motor actuators for the *simulated e-puck variant 2*; and 8 distance sensors for the *real e-puck*. The size of the readout output layer (n_{o}) is equivalent to the number of predefined locations or rooms in the environment. The postprocessing function g for the readout units is given by (2.21) in Section 2.7. The resampling rate ($d_t = 50$) and leak rate ($\alpha_1 = 0.6$) are found by performing a grid search for the SINAR model. The experiments with the simulated and real e-puck consider 3 neural pools in the reservoir, each one with different leak rates ($\alpha_1, \alpha_2, \alpha_3$). These leak rates enable the operation of the reservoir in multiple different timescales, which is an important feature when the task considers a robot with a varying speed (in our case, the robot can have 3 different velocities). The downsampling rate d_t and the three leak rates were empirically chosen for both e-puck models.

Table 4.6: Parameter configuration for room detection

Model	sim.e-puck variant 2	real e-puck
Number of input channels	$n_i = 10$	$n_i = 8$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.3$
Input scaling	$\upsilon_i^r = 0.9$	$\upsilon_i^r = 2$
Input downsampling	$d_t = 20$	$d_t = 1$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 800$	$n_r = 1200$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate 1	$\alpha_1 = 0.05$	$\alpha_1 = 0.005$
Leak rate 2	$\alpha_2 = 0.8$	$\alpha_2 = 0.01$
Leak rate 3	$\alpha_3 = 1$	$\alpha_3 = 0.2$
Number of output channels	$n_o = 4$	$n_o = 4$
Output feedback to reservoir	no	no

4.4.5 Experimental Results

4.4.5.1 Pose estimation

The results on test data can be seen in Fig. 4.11. The first plot shows that the RC system can cope very well with the long T-maze. The robot trajectory is not completely repetitive: between timesteps 200 and 400, the robot captures two targets (observe the changes on Y coordinate) whereas between timesteps 800 and 1200, four targets are captured (the Y robot coordinate fluctuates for a longer period). The average NMSE over 30 randomly generated reservoirs is 0.025 with standard deviation of 0.02.

The second plot shows the performance on test data for environment S5. This second experiment required more parameter tuning than the first one due to the apparent increased complexity associated with the robot trajectory (see Fig. 4.11). Nonetheless, the RC network correctly follows the target robot's pose (x,y and heading). The average NMSE over 30 runs is 0.283 and the standard deviation is 0.072.

These experiments show that RC is able to create an implicit map of the environment and use it for localization purposes on a

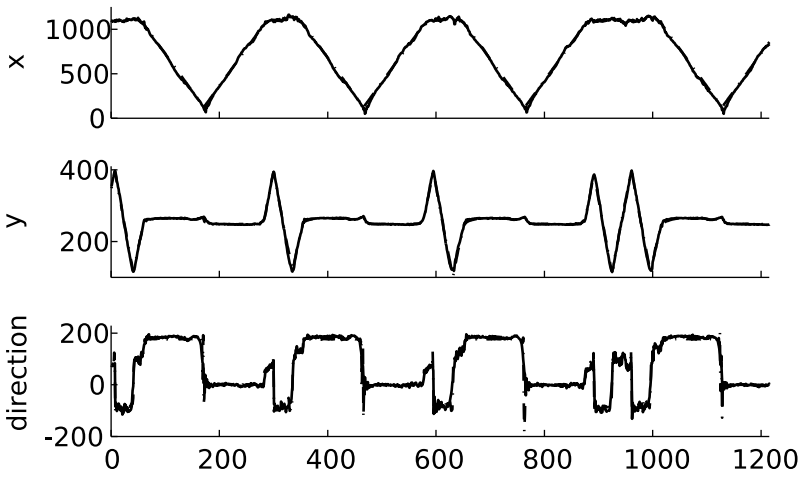
coordinate basis. It works in environments that look spatially very similar and need short-term memory to know the correct location, and even in complex maze-type environments.

4.4.5.2 Location detection

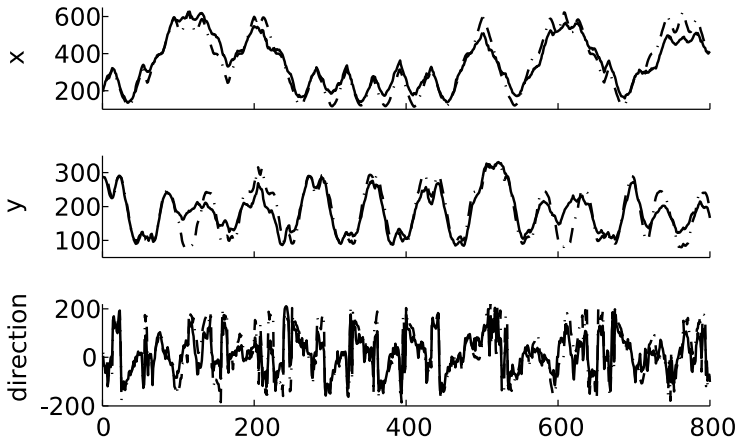
The next experiment is accomplished with the first environment S6 from Fig. 4.8 and lasts 180.000 timesteps. The resulting robot occupancy grid can be seen in Fig. 4.12(a): it shows that the reservoir predicts the current location of the robot very well, with a classification rate of 91.6% on test data. Another experiment uses the same environment S6 with 11 additional slow moving obstacles distributed throughout the environment. These dynamic objects alter the behavior of the robot, e.g., by blocking navigation and producing avoidance behaviors, resulting in an extra source of noise to sensory readings. The simulation takes 180.000 timesteps. The respective occupancy grid in Fig. 4.12(b) shows that the RC network correctly predicts the location in 81.1% of the samples. Some of the mispredictions are located a bit further from the actual position, due to the new source of dynamics and noise, although they generally tend to be very short. As comparison, the RNN-based room detector in Forster et al. (2007) which has 36 inputs coming from a laser range scanner presents a test classification rate of 81.8% for non-noisy environment and 82.8% with 10 slow moving obstacles for a simulated house environment of 15 rooms.

The experiment accomplished in environment S7 from Fig. 4.8(b) tackles sensor aliasing problems more visibly: the environment has many symmetric areas and trajectories. For instance, going from position 27 to 26 looks identical to the robot as going from position 22 to 24. The simulation takes 150.000 timesteps. The resulting occupancy grid in Fig. 4.12(c) shows that a classification rate of 89.1% on test data makes the RC network an efficient location detector in symmetric environments.

The following experiments are based on the simulated e-puck variant 2 robot in environment E2. The results in Fig. 4.13 show that the performance increases significantly when a reservoir with 3 neural pools of distinct leak rates is used instead. Several mispredictions present in Fig. 4.13(a) are not present anymore in Fig. 4.13(b) which corresponds to the multiple leak rates case. The increase in

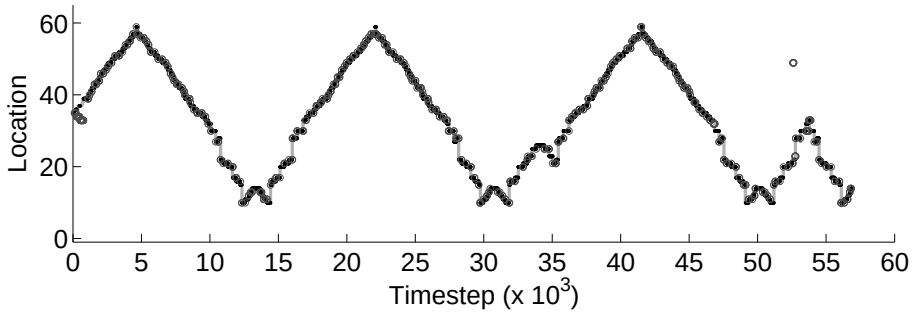


(a) Pose estimation for S4

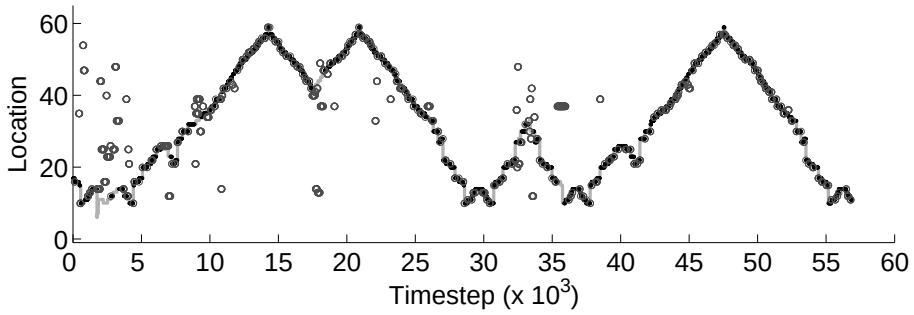


(b) Pose estimation for S5

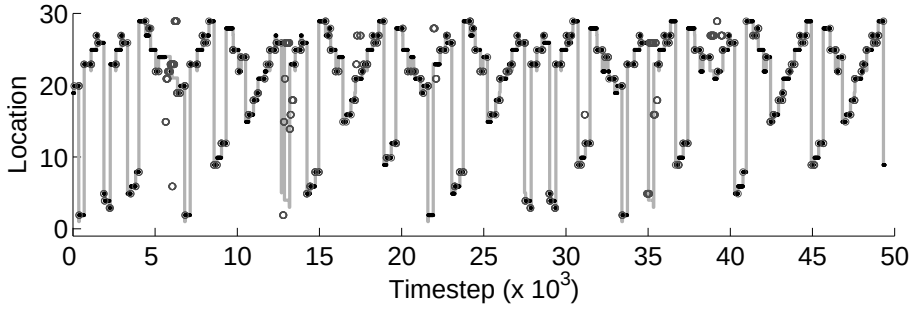
Figure 4.11: Pose estimation on test data using the SINAR model. The dashed lines represent the desired pose (i.e., robot's coordinates and heading) whereas the solid line corresponds to the output of the RC network. (a) plot for environment E4, the long T-maze (dashed lines not visible). (b) plot for environment E5, a diverse and dynamic environment.



(a) Location Detection in S6



(b) Location Detection in S6 with 11 moving obstacles



(c) Location Detection in S7

Figure 4.12: Robot occupancy grids showing the predicted and the true robot location as black points and a solid gray line, respectively, on test data. Mispredicted locations are represented by a circle. (a) Experiment in environment S6 with a test performance of 91.6% of correct detection. (b) Experiment in environment S6 with 11 slow moving obstacles with a test performance of 81.1% of correct detection. (c) Experiment in environment S7 with a test performance of 89.1% of correct detection.

Table 4.7: Performance results for location detection

Model (Env)	Leak	Timesteps	Train Perf.	Test Perf.	Std Test
SINAR (S6)	no	180 K	94.0 %	**89.2 %**	0.4%
SINAR (S6)	yes(1)	180 K	94.3 %	**90.7 %**	0.1%
SINAR (S7)	no	150 K	94.4 %	**88.6 %**	0.3%
SINAR (S7)	yes(1)	150 K	94.4 %	**89.1 %**	0.3%
SINAR ($S6^{mov}$)	no	180 K	93.0 %	**76.0 %**	0.7%
SINAR ($S6^{mov}$)	yes(1)	180 K	94.2 %	**79.1 %**	0.5%
sim.e-puck variant 2 (E2)	no	40 K	87.2 %	**78.9 %**	0.4%
sim.e-puck variant 2 (E2)	yes(3)	40 K	90.4 %	**85.1 %**	0.5%
real e-puck (E3)	yes(3)	192 K	–	**83.3 %**	–

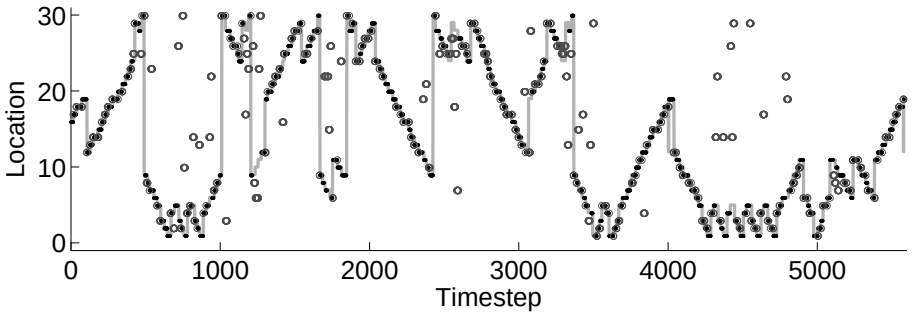
performance is of at least 6% in correctly classified samples when compared to the experiment with a normal reservoir, i.e., without multiple leak rates.

A summary of the localization experiments with corresponding results is shown in Table 4.7. This table also presents results from experiments which only used dataset resampling, thus, disregarding the use of leak rates (indicated by the word *no* in the column *Leak*). Each experiment is evaluated 30 times with different stochastically generated reservoirs and the results are averaged over these 30 runs. One can observe that the use of leak rates in addition to dataset resampling yields the highest increases in performance for the experiments with the SINAR model in environment $S6^{mov}$ (with moving obstacles), i.e., 3.1% increase, and with the simulated e-puck variant 2 robot, using 3 neural pools of distinct leak rates, i.e., 6.2% increase.

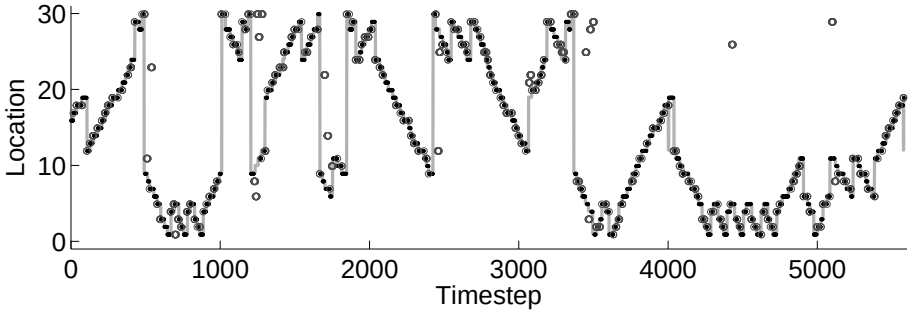
Experiments only considering distance sensors as input, by removing actuator feedback, result in similar performance reported for the experiments reported in this section.

4.4.5.3 Room detection

The results for environment E2 are shown in Fig. 4.15 using the configuration from Table 4.6. The RC-based room detector is very efficient for at least 7000 timesteps, showing a performance of 93.6% on test data. There are few mispredictions, and most of them are



(a) No leaky-integrator units (or $\alpha = 1$)



(b) 3 neural pools of distinct leak rates

Figure 4.13: Location detection performed by a RC network for the simulated e-puck variant 2 robot in environment E2 on test data. The gray solid line represents the actual location whereas the black points are the predicted location. Mispredictions are marked with circles. (a) A normal reservoir gives a performance of 77.7%. (b) A reservoir containing 3 neural pools of distinct leak rates yields a performance of 84%.

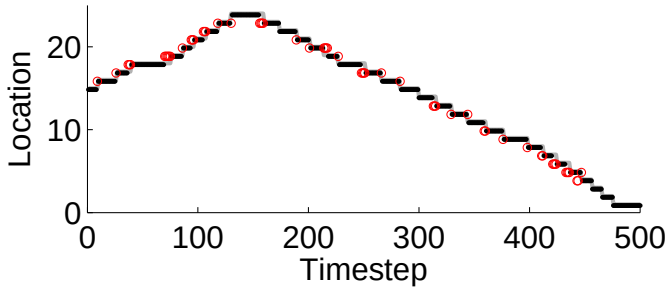


Figure 4.14: Location detection results on test data using the real e-puck robot in environment E3. The gray solid line represents the desired robot room while the actual network output is represented by black points (the miss-classifications are marked with red circles).

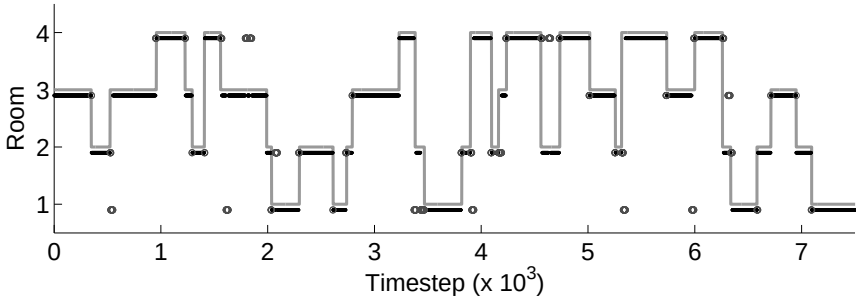


Figure 4.15: Room detection performed by an RC network for the e-puck robot in environment E2 on test data. The performance in terms of correctly classified rooms is 93.6%. The gray solid line represents the actual room whereas the black points are the predicted room. Mispredictions are marked with circles.

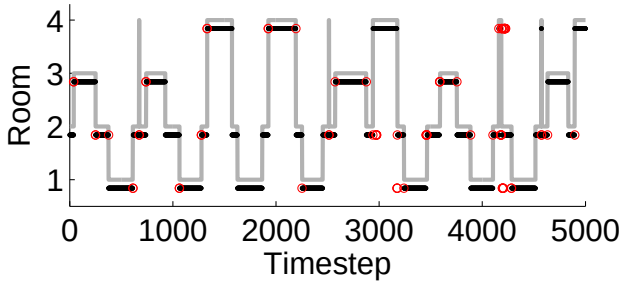


Figure 4.16: Room detection results on test data using the real e-puck robot in environment E3. The gray solid line represents the desired robot room while the actual network output is represented by black points (the miss-classifications are marked with red circles).

located in the temporal boundaries between one room and the next one.

The results for the real e-puck in environment E3 are shown in Fig. 4.16. The training process uses 7/8 of the data (1/8 is used for testing) and it takes around 208 seconds, of which 140 seconds are spent on generating the matrix $\mathbf{X}$ with the harvested reservoir states[4] (see Section 2.5).

The classification performance for room detection is very good considering the random behavior of the robot: 98.1% of the test data is correctly classified. Fig. 4.16 shows that the trained RC network is an efficient room detector and that most errors are made

[4]Considering an Intel Core2 Quad CPU platform with 8 GB RAM.

Table 4.8: Performance results for room detection

Model (Env.)	Timesteps	Train	Test	Test Perf. Rooms 1, 2, 3 and 4 resp.			
sim.e-puck variant 2 (E2)	39 K	97.2%	**93.1%**	**95.4%**	**91.5%**	**91.8%**	**93.1%**
real e-puck (E3)	192 K	–	**98.1%**	–			

in the boundaries between a room and the next one, represented by red circles in the figure.

Table 4.8 shows statistics for the room detection task. Each one of the rooms in E2 is correctly detected by a mean rate of at least 91%. The mean classification rate on test data for the real e-puck robot in environment E3 is 98.1%. For comparison, the RNN-based room detector from Forster et al. (2007) correctly detects rooms 80% of the time, considering 35 inputs from a laser range scanner and a simulated environment with 15 rooms.

4.4.5.4 Kidnapping

The RC network also copes well with the kidnapping situation (also reported in (Forster et al., 2007)). In a new experiment using environment S6, the robot is replaced from location 34 to location 20 (see Fig. 4.17(a)). The network is able to successfully predict the robot position when the robot reaches location 16. Note that the predicted locations following the kidnapping are near location 34, as if the robot kept the original path. After some timesteps, the network *realizes* it is actually in location 16, given the history of sensory inputs since the kidnapping. Another example is given in the second plot of Fig. 4.17(a), where the robot is kidnapped from location 41 to location 46. Note that the RC network is not trained for the kidnapping event. At first, the performance of the kidnapping situation reported above is similar to the work in (Forster et al., 2007): both can predict the correct location (room in their work) after some timesteps, despite the different robot model and environment used in (Forster et al., 2007); for instance, they use 36 inputs while the e-puck robot has only 8 distance sensors.

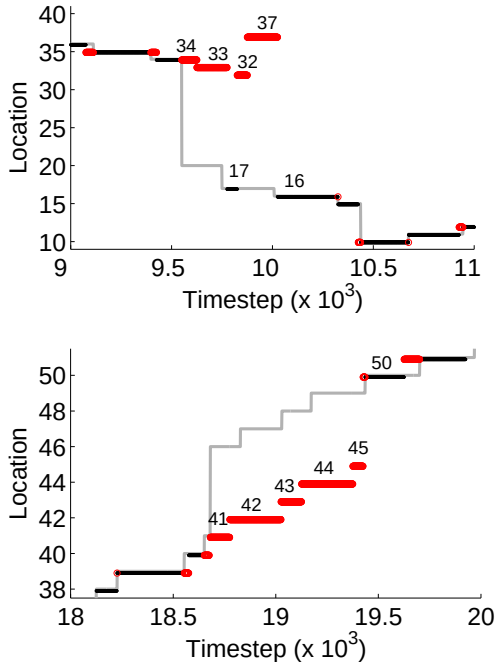


Figure 4.17: Occupancy grid after kidnapping the robot in environment S6 of Fig. 4.8. The solid gray line represents the actual robot position. Correct predictions are given by black points while wrong predictions are marked with circles. The predictions of the RC network are labeled with numbers after the robot is kidnapped. (a) At time step 9551, the robot is moved from position 34 to position 20. The reservoir network predicts successfully the current robot position when the robot is in location 16. The robot visits 2 locations (20 and 17) until the successful prediction. (b) At time step 18730, the robot is moved from position 41 to position 46. The reservoir network predicts successfully the current robot position when the robot is in location 50. The robot visits 4 locations (47, 48 and 49) until the successfull prediction.

4.5 Conclusion

This chapter has shown that an RC network is a **powerful black-box modeling method for learning implicit models of the environment of a mobile robot from local noisy sensory information**. The reservoir's recurrent connections and its dynamic regime provide a mechanism of short-term memory which is fit to solve sensor aliasing problems, where multiple locations that the robot may be in the environment provide identical readings in the

local low-dimensional sensory space. With the reservoir's fading memory and its non-linear projection in a high-dimensional space, sensory readings are correctly mapped to locations (or events) in the readout output layer as the mobile robot navigates in complex, dynamic, and symmetric environments. The experiments are carried in several environments from two different simulators and using the real e-puck robot in an unstructured environment.

It has been shown that the same reservoir-based method can model **position detectors at different levels of abstraction**. The continuous-valued robot's pose in the global coordinate frame, given by x, y coordinates and heading, can be modeled by an RC network with feedback from the output units and trained to perform regression. More abstract discrete representations of the environment such as small delimited locations and whole rooms can also be learned with an RC network via classification. Thus, Reservoir Computing has shown to be a general method for modeling environmental representations of different types and scales, considering noisy sensors which provide only local, ambiguous information from the environment.

The **importance of timescales in RC-based systems** is clearly demonstrated in this work, especially when the speed of a mobile robot is not constant. By employing multiple leak rates organized in pool of neurons in the reservoir, the resulting dynamics of this reservoir better fits the multiple timescales components of the input signal given by the distance sensors of a mobile robot. From an RC perspective, performance could further be improved, for example, by using a so called band-pass reservoir (wyffels et al., 2008). This idea consists of adding band-pass filters to a reservoir so that it can capture a wide range of frequencies or timescales. This setup could greatly improve the performances on tasks which consider a wide range of robot speeds.

The reservoir, however, also models the underlying behavior or navigation attractor given by the robot controller. This controller generates training data according to a reactive sensory-motor mapping, perturbed by noise and some stochastic decisions. When this data is used for training an RC network, the locations are learned based on the projection of this navigation attractor in sensory space to the dynamical reservoir's space. As a consequence, locations

which are never visited in the training dataset will not be recognized by the trained RC network. Similarly, an RC network trained with data from a specific controller could not generalize to a different controller in the testing stage. This is evident as one notes that the training was accomplished based on the *projection of the navigation attractor*. An ultimate consequence of this is that there must be important limitations to using this architecture as a general localization system for mobile robots. Particularly, **the use of the RC architecture is limited to test data generated from controllers which are similar to the ones used to collect training data**.

By learning an implicit sensor-based representation of a complex robot environment, an RC network can be used to boost navigation capabilities of an intelligent system. This network, by predicting the current location of a robot and even the previously visited location, can convert a simple reactive navigation system (from Chapter 3) to a goal-directed navigation system capable of generating a sequence of reactive behaviors through a multi-room environment until a destination room is achieved. This is the focus of the next chapter.

5
Goal-directed Navigation

In order to learn context sensitive sensory-motor coupling via a bottom-up approach, it is necessary to design architectures which model different aspects of a navigation task in distinct timescales. As seen in previous chapters, RC networks can be used to perform localization as well as to learn multiple navigation behaviors in particular environments. In this chapter, by combining previous results, a hierarchical architecture composed of two modules is proposed: a localization module and a navigation module which operate at slow and fast timescales, respectively. The former module is trained to predict the current and the previously visited room based on the current distance sensors' readings, whereas the latter is trained to steer the robot in a goal-directed manner based on the input signals received from the localization module, distance sensors, and the target room. After training this multiple timescale hierarchical architecture with examples of navigation routes in simulated environments, the resulting RC-based controller is able to successfully navigate to specific target rooms in both simple and large unknown environments composed of many rooms.

5.1 Introduction

The RC networks for modeling navigation attractors or behaviors presented in Chapter 3 are based on an external switching mechanism modeled explicitly by an additional input to the network. Although these networks can model multiple behaviors in an envi-

ronment, their applicability is limited to small environments.

This chapter tackles the problem of goal-directed robot navigation based on implicit learned maps in large unknown environments. In this context, the task of a mobile robot is to reach a well defined target room (indicated by a human user) in a particular environment. Traditional approaches would require the explicit modeling of the environment and expensive path planning routines to be executed on the model.

Instead of explicit modeling the environment, this work follows a bottom-up approach in that hierarchical RC networks are trained to perform goal-directed navigation according to a supervised learning process as presented in Chapter 3. While Chapter 3 has dealt with learning of sensory-motor coupling without internal models or representations of the environment, this chapter combines the RC-based location detector developed in Chapter 4, as a form of internal model predicted solely from current sensory readings, with the supervised learning process elaborated in Chapter 3.

The proposed hierarchical architecture is composed of two reservoir modules, one for localization and another for navigation. The localization reservoir receives input from only 8 low-accuracy distance sensors and predicts the current and previously visited robot room. The mapping between reservoir states to the predicted rooms is learned in a supervised way from examples of robot trajectories in the considered environment (as shown in Chapter 4). In a second learning stage, the navigation reservoir is trained with several examples of routes from a starting location to a goal location, using information from the localization reservoir (i.e., predicted locations), the distance sensors and the desired goal location, given as input to the navigation system. Therefore, this navigation module integrates different types or *modalities* of input and can simultaneously learn reactive and deliberative behaviors (also shown in Chapter 3 for delayed response tasks like the T-maze task).

It is important to observe that the internal reservoir memory is essential for learning an internal model of the environment and of the robot task. The recurrent reservoir has states which reflect the recent history of inputs, representing a short-term memory capable of combining and dealing with different sources of information. Its inherent capability for modeling temporal non-linear systems makes

it very interesting for constructing internal models. As argued by J. Tani (Tani, 2007), *behavior-based systems without internal models are blind*. So, by using reservoir computing techniques for modeling goal-directed navigation tasks, the aim is to create an unifying and efficient method for supervised learning of deliberative and reactive behaviors as well as its underlying internal models.

Chapter 3 has shown that multiple behaviors can be embedded into a single dynamical reservoir and switching between two behaviors is achieved by an extra input channel which shifts the operating point of the reservoir state space. This transition initiated by an external input is able to change the sensory-motor coupling to a different operation mode. These different operation modes are called reactive *navigation attractors* or behaviors. In this chapter, the internal models learned with RC by the localization module represent hidden layers whose activity also induces changes in the sensory-motor coupling of the navigation reservoir. Thus, the localization module guides the operation of the navigation module, by changing the sensory-motor coupling into different navigation attractors as the robot enters and leaves rooms. In this way, the proposed architecture is able to perform a sequence of primitive reactive behaviors which lead the robot to the desired destination in the environment.

Although the proposed model does not use any form of path integration (e.g., odometry), the reservoir provides a short-term memory of previous inputs, making it possible that the robot maintains an estimate of its current location for additional timesteps even in the absence of sensory input.

The prominent characteristics of the current approach are threefold: no special environment landmarks are required; it works for small mobile robots having few low-accuracy distance sensors; and deliberative and reactive navigation components are learned with examples of the desired sensory-motor coupling with the same hierarchical architecture.

The experiments are accomplished with a simulation model of the e-puck robot extended with longer-range ($[5 - 80]$ cm) infrared sensors in the Webots environment. It is shown that proposed system can learn with examples to drive a robot to a desired goal location in simple and bigger (9 rooms) environments using only 8 low-accuracy sensors and the goal location as input.

5.2 Related Work

In Tani and Nolfi (1999), a self-organized way of learning hierarchies through competition of RNN experts is presented. Each RNN in a level competes to model part of a reactive behavior during robot navigation, so that different RNNs model different temporal segments of the sensory-motor flow. Higher level networks are trained to predict the gate openings from lower-level networks in a slower timescale. While its training is done online via gradient descent, our approach uses offline linear regression (keeping the recurrent part of the network fixed). Moreover, it only models the behavior by a bottom-up process, that is, it can predict the next sensory input but can not generate control actions in contrast to our RC-based navigation architecture.

The localization reservoir in our proposed architecture has binary outputs representing the context of the robot (where the robot came from and where it is currently located). In Tani (2003), the parametric bias also changes in a step-wise (binary) way, and is the sole information shared by both levels in the hierarchy. The functional and structural aspect of our proposed hierarchical architecture is similar to the one presented in Tani (2003) in that a higher-level network (corresponding to our localization network) guides the sequential execution of low-level motor primitives in the lower-level network (corresponding to our navigation network). The main differences is that his method is applied to a robotic arm and training the architecture is based on back propagation through time (BPTT). Furthermore, our approach for learning navigation behaviors is based on the concept of *sub-space attractors* which embeds primitive reactive behaviors into linearly separable high-dimensional sub-spaces.

5.3 Methods

5.3.1 Extended E-puck Robot Model

The e-puck robot is described in Section 4.2.1 or Appendix B.2. The *variant robot model* used in the following experiments is the **simu-**

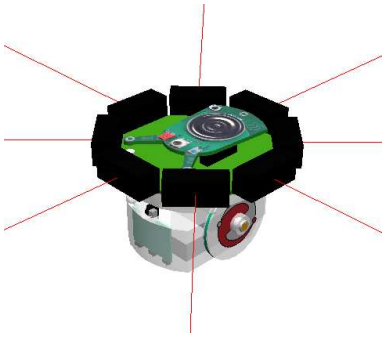


Figure 5.1: *Modified e-puck robot* from Webots simulation environment, **extended** with simulated longer-range infra-red sensors capable of reading distances from 5 cm to 80 cm.

lated e-puck extended with 8 infra-red sensors which can measure distances in the range [5-80] cm. The original simulation model of the e-puck has a 5.20 cm diameter, but it increases to 10 cm when modified with the extra turret for the infra-red sensors. The speed of the robot is limited to the interval $\pm[0, 300]$ steps/s (or $\pm[0, 3.77]$ cm/s).

While the e-puck robot navigates in its environment using the Webots simulator (Michel, 2004), a dataset containing sensory readings, actuators (motor speeds), and current and previously visited rooms is recorded into a Matlab environment (by process communication implemented with TCP/IP sockets). The robot controller used to collect data is described in the following section.

5.3.2 Learning to Navigate to Goals by Demonstration

The supervised learning procedure consists of two stages as follows.

Data Generation and Collection In this stage, several examples of routes through the environment are generated, in which the robot navigates from a starting room to a destination room according to a predefined algorithm which uses *primitive reactive behaviors* to steer the robot in different modes. All required data for training are collected during this stage such as: distance sensors and destination room (which will be used as input channels); and the currently and previously visited

robot rooms and desired motor actuators (for desired hidden or output units).

Training The second stage involves the training of the RC networks with the data generated in the first stage. Afterwards, the trained RC-based navigation system can be used to drive the robot to specific destination rooms given as input.

To actually generate examples of navigation routes, two primitive reactive behaviors or navigation attractors are used to steer the robot through different paths inside a room. They are called *Left attractor* and *Right attractor*. Fig. 5.2 shows how these primitive behaviors can be used in sequence to generate complete paths to a destination room in an hypothetical environment. As a matter of simplicity, both primitive behaviors are implemented by different Braitenberg vehicles (Braitenberg, 1984), whose motors' outputs consists of a linear combination of the current sensory readings (i.e., a linear sensory-motor mapping). The Braitenberg vehicle which avoids obstacles more intensely at the left side than at the right side forms a reactive **Left navigation attractor**. The **Right navigation attractor** is constructed in a similar way. These primitive behaviors form **spatial attractors** since they tend to follow *cyclic sensory-motor patterns in space* in static environments.

In the dynamical system space of the reservoir, *sub-space attractors* are formed resulting from the sensory-motor coupling which is learned with data collected using the two primitive behaviors. In other words, the reservoir should learn to reproduce the same context-dependent sensory-motor coupling, where each context transition (entering a room through a specific door) causes a change in the sensory-motor coupling (or navigation attractor). As the reservoir-based navigation system is tightly coupled with the environment, spatial navigation attractors once projected into the dynamical system space can be seen as sub-space attractors shifted by internal and/or external context switches. Fig. 5.3 shows the corresponding left and right sub-space attractors in a simplified bidimensional dynamical system space for the sequence of spatial navigation attractors shown in Fig. 5.2. Starting at room 1, the robot gets an external input for the goal destination, indicated by the transition given by the dashed arrows, and performs a series of primitive

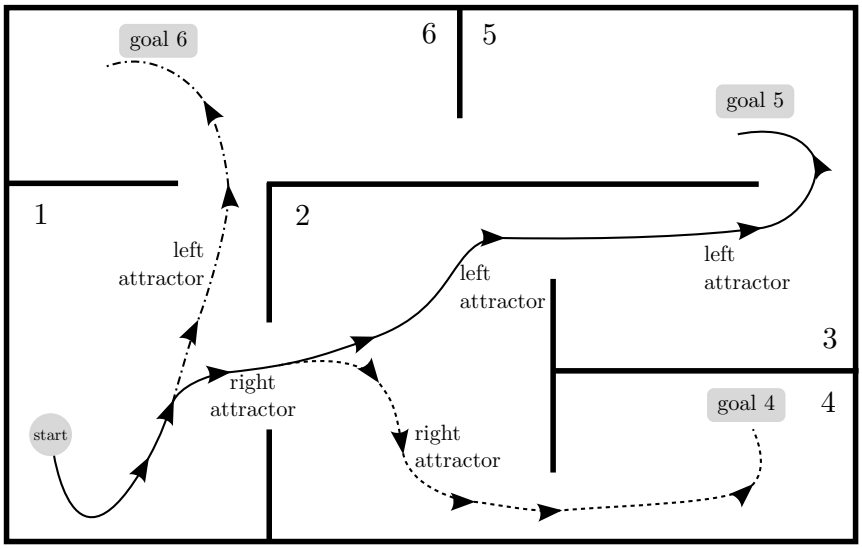


Figure 5.2: Example of goal-directed navigation as a sequence of reactive *navigation attractors* or behaviors: *left attractor* and *right attractor*. The plot shows an hypothetical environment with 6 rooms and robot trajectories represented by solid and dashed lines, with arrows indicating the orientation of the robot. The two simple reactive behaviors, i.e., left and right attractors, lead the robot to different paths in a room. Three different trajectories leading to goals 4, 5 and 6 are shown in the environment. For instance, the mobile robot reaches goal 5, starting at room 1 and choosing: right attractor, left attractor and left attractor. Examples of routes like these are generated for the supervised learning process.

behaviors which are fired by internal transitions, represented by solid arrows, which ultimately lead to the final destination. For instance, the transition r.2 g.5 signals that the robot entered room 2 from room 1 while its destination (goal) is room 5. These internal transitions will be modeled by a *localization* reservoir, which predicts the current and previously visited room. The *navigation* reservoir will model the sensory-motor coupling given by navigation attractors, whose operation is modified by the guidance of the localization reservoir. These two RC networks form an hierarchical architecture described in the following section.

5.3.3 Hierarchical RC Architecture

The Hierarchical Reservoir Computing (HRC) controller is composed of two RC networks or modules: the localization and the

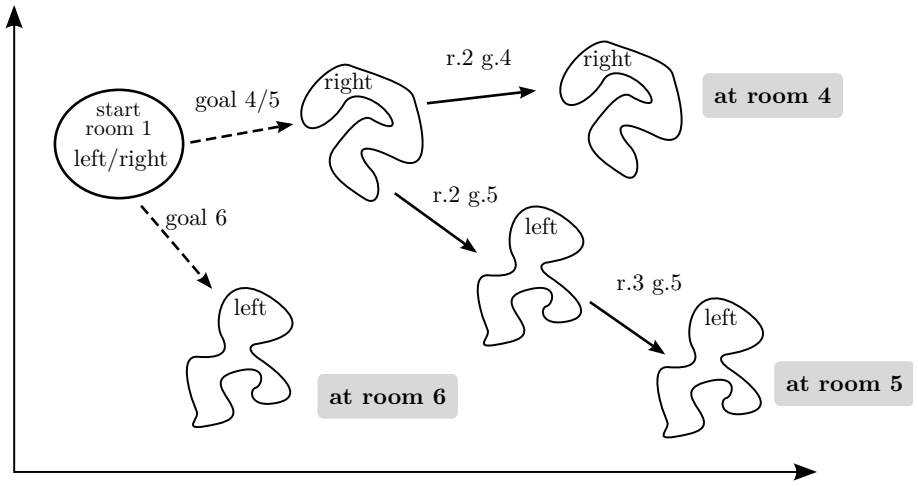


Figure 5.3: Simplistic view of navigation attractors in bi-dimensional dynamical system space corresponding to the routes to goals 4, 5 and 6 shown in Fig. 5.2. The circle represents the starting position of the robot, which can be in left or right attractor. Dashed lines represent transitions between sub-space attractors in the dynamical system space given by external input channels, while solid lines indicate transitions given by internal hidden activity, resulting from the internal predictions of the current and possibly the previously visited location, for instance (the transition r.2 g.4 is an abbreviation of *room 2* and *goal 4*, i.e., the robot is located at intermediate room 2, with room 4 as final destination). The goal rooms are reached after a sequence of sub-space attractors, representing simple reactive behaviors, has been performed.

navigation modules (see Fig. 5.4). It is relevant to observe that the localization reservoir operates at a much slower timescale than the navigation reservoir since transitions between rooms are very sporadical, requiring a reservoir with slow dynamics (achieved by using a low leak rate α) when compared to the required quick reaction of reservoirs implementing navigation behaviors.

The learning process is divided in two stages:

1. The **localization module** is trained with examples of robot trajectories to detect the current and previously visited robot room using the controller described in last section.

2. Then, the **navigation module** is trained with new examples of robot trajectories, **but now using the prediction of the trained localization module as input**.

By rewriting equations (2.9) and (2.2) for the localization module, we get:

$$\begin{aligned} \mathbf{x}_{\text{loc}}[n+1] = \quad & (1-\alpha_{\text{loc}})\mathbf{x}_{\text{loc}}[n] + \alpha_{\text{loc}} f((\mathbf{W}^{\text{r}}_{\text{i\_loc}}\mathbf{u}_{\text{dist}}[n] + \\ & \mathbf{W}^{\text{r}}_{\text{r\_loc}}\mathbf{x}[n] + \mathbf{W}^{\text{r}}_{\text{b\_loc}})), \end{aligned} \tag{5.1}$$

$$\mathbf{y}_{\text{c}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{c}}\mathbf{x}_{\text{loc}}[n+1]), \tag{5.2}$$

$$\mathbf{y}_{\text{p}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{p}}\mathbf{x}_{\text{loc}}[n+1]), \tag{5.3}$$

where $\mathbf{y}_{\text{c}}$ and $\mathbf{y}_{\text{p}}$ are vectors of size n_l representing the predicted *current* and *previous* robot locations, respectively; n_l is the number of locations or rooms in the environment and $g(\mathbf{x})$ is a winner-take-all function which gives $+1$ for the highest input and -1 otherwise. The other parameters and variables have the same meaning as the ones in Section 2.2, but have new subscripts for identifying the localization reservoir.

Analogously, the equations for the navigation module are as follows:

$$\begin{aligned} \mathbf{x}_{\text{nav}}[n+1] = \quad & (1-\alpha_{\text{nav}})\mathbf{x}_{\text{nav}}[n] + \alpha_{\text{nav}} f((\mathbf{W}^{\text{r}}_{\text{i\_nav}}\mathbf{u}_{\text{multi}}[n] \\ & + \mathbf{W}^{\text{r}}_{\text{r\_nav}}\mathbf{x}[n] + \mathbf{W}^{\text{r}}_{\text{b\_nav}})), \end{aligned} \tag{5.4}$$

$$\mathbf{y}_{\text{nav}}[n+1] = \quad g(\mathbf{W}^{\text{out}}_{\text{nav}}\mathbf{x}_{\text{nav}}[n+1]), \tag{5.5}$$

where $\mathbf{y}_{\text{nav}}$ is a vector with the speeds for the left and right wheels of the robot; and $\mathbf{u}_{\text{multi}}(t)$ is a concatenated input vector consisting of

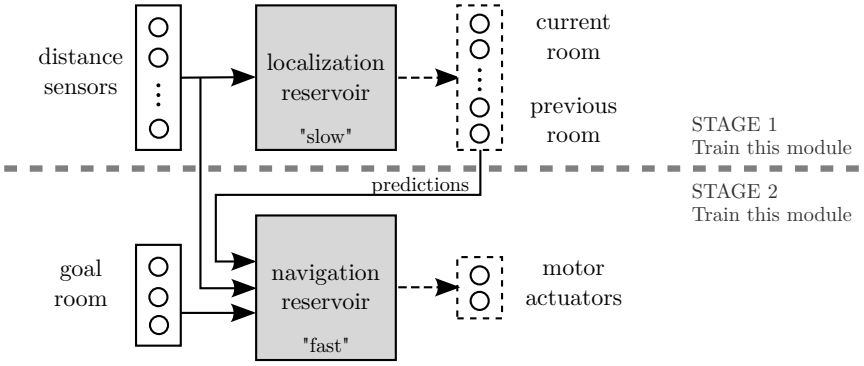


Figure 5.4: Hierarchical architecture with localization and navigation modules. The navigation and localization reservoirs are randomly generated recurrent networks which are not trained, but left fixed. Trainable components (or weights) are shown in dashed lines. The sensory input feeds both reservoirs, being mapped to a high-dimensional space, where learning occurs. The navigation reservoir receives input also from the localization module and the target location and outputs the desired motor actuators. Stage 2 trains the navigation module using the **predictions** given by the localization module, trained in Stage 1.

the distance sensors, the current and previous predicted locations, and the goal location

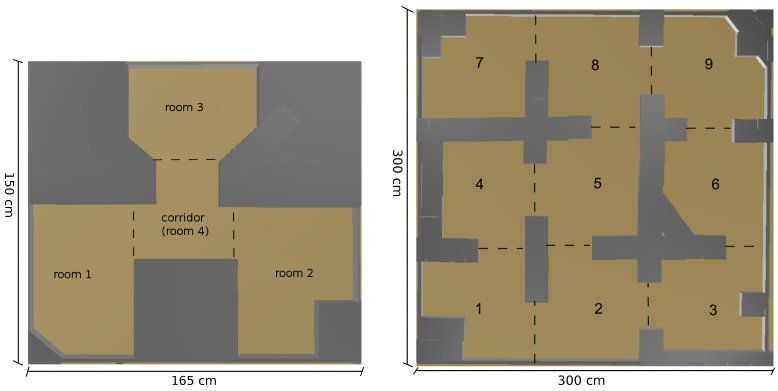
$$\mathbf{u}_{\text{multi}}(t) = [\mathbf{u}_{\text{dist}}^T(t)\mathbf{y}_{\text{c}}^T(t)\mathbf{y}_{\text{p}}^T(t)\mathbf{u}_{\text{goal}}^T(t)]^T.$$

The weight matrices $\mathbf{W}_{\cdot}^{\text{out}}$ in Equations (5.2), (5.3) and (5.5) are trained using linear regression as explained in Section 2.5. All other weight matrices connecting to the reservoir are randomly generated at the beginning of the experiment and left fixed.

5.4 Experiments

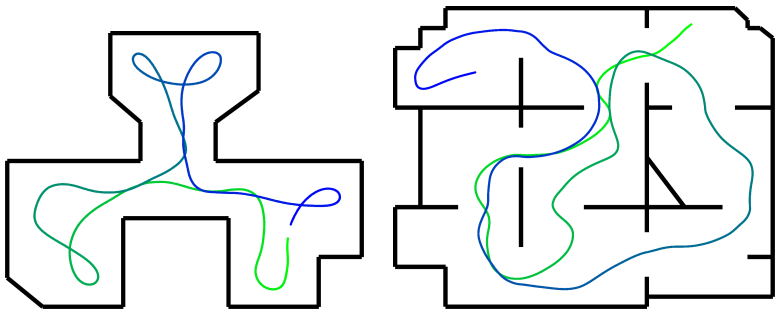
The proposed HRC architecture was evaluated in two environments. Environment E4 is composed of three rooms connected by a central corridor (see Fig. 5.5). A second, larger environment E5 is made of 9 rooms with open doors connecting them.

For the first environment, there are two training datasets, one consisting of 500.000 samples (4 hours and a half of simulation time) for training the localization module in a first step and the other one consisting of 100.000 samples for training the navigation reservoir in



(a) E4 (b) E5

Figure 5.5: Webots environments used for experiments. (a) Environment (165 cm × 150 cm) with 3 goal rooms and a connecting corridor. (b) Large environment (300 cm × 300 cm) with 9 rooms (goal rooms are 1, 3, 7 or 9). Dashed lines represent boundary limits between rooms.



(a) E4 (b) E5

Figure 5.6: Samples of robot trajectories used as training examples for the HRC controller. (a) Trajectory in E4. (b) Trajectory in E5.

a second step. These training datasets contain examples of trajectories of a robot continuously going from an initial room to a target room (see Fig. 5.6(a) for an example) - there are 6 possible routes in environment E4.

The second environment E5 has 9 rooms and only 4 of them will be used as starting and goal locations: rooms 1, 3, 7 and 9. In this way, starting in one of the 4 locations, there are 12 possible shortest (optimal) routes that the robot can follow. The training

Table 5.1: Parameter configuration for Environment E4

Module	Localization	Navigation
Number of input channels	$n_i = 8$	$n_i = 19$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 1$	$\upsilon_i^r = 1$
Input downsampling	$d_t = 10$	$d_t = 5$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 400$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) = 0.98$	$\rho(\mathbf{W}_\mathrm{r}^\mathrm{r}) = 0.98$
Leak rate	$\alpha = 0.01$	$\alpha = 1$
Number of output channels	$n_o = 8$	$n_o = 2$
Output feedback to reservoir	no	no

datasets are also generated in the same way as before, but now 500.000 samples represent only 32 routes, which are less examples for training than for environment E4. See Fig. 5.6(b) for an example of robot trajectories generated with the supervisor controller.

5.5 Settings

For both environments E4 and E5, the two datasets of 500.000 and 100.000 samples were downsampled by a factor of $d_t^{\mathrm{loc}} = 10$ and $d_t^{\mathrm{nav}} = 5$ respectively (values empirically chosen to give best performance), resulting in two new datasets of 50.000 and 20.000 samples for training the localization and the navigation module, respectively. As these sampling rates are different from each other, signals from the localization reservoir $\mathbf{y}_\mathrm{c}$ and $\mathbf{y}_\mathrm{p}$ are upsampled to the same sampling rate of the navigation reservoir before they are used as input to that module.

The parameter configuration is given in Table 5.1 for environment E4 and Table 5.2 for environment E5. Some of these parameters are described in Chapter 2. At it can be seen from these tables, the experiments on both environments use the same parameter configuration, except for the number of outputs n_o of the localization

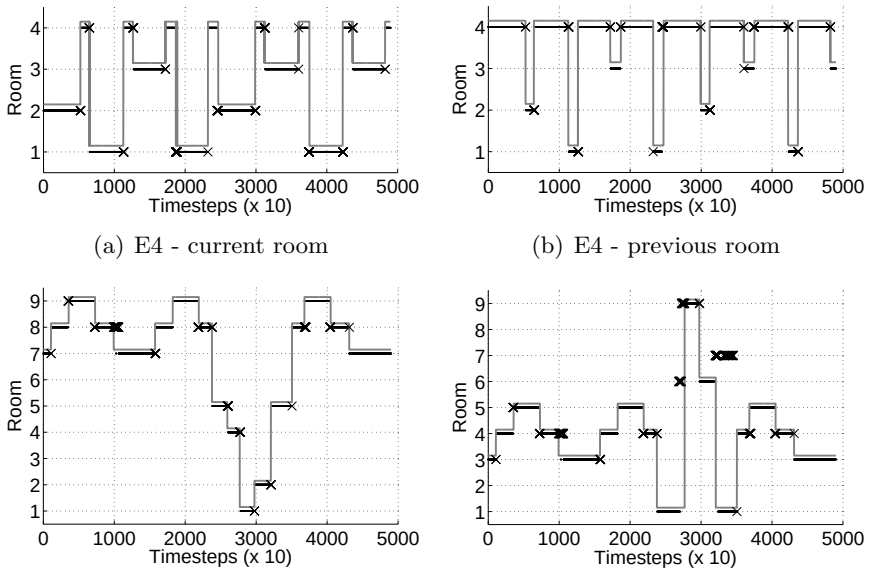
Table 5.2: Parameter configuration for Environment E5

Module	Localization	Navigation
Number of input channels	$n_i = 8$	$n_i = 30$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 1$	$\upsilon_i^r = 1$
Input downsampling	$d_t = 10$	$d_t = 5$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 400$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.98$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.98$
Leak rate	$\alpha = 0.01$	$\alpha = 1$
Number of output channels	$n_o = 18$	$n_o = 2$
Output feedback to reservoir	no	no

module, and the number of inputs n_i for the navigation reservoir. For environment E5, $n_o^{\mathrm{loc}} = 18$ (9 units for previously visited room and 9 for the current room) and $n_i^{\mathrm{nav}} = 30$ (18 from the localization module + 4 goal inputs + 8 distance sensors). Parameters α and d_t were found by a grid search in the case of the localization module (offline testing), and empirically in the case of the navigation module (online testing by trial and error).

5.6 Results

The localization performance on test data, consisting of 50.000 samples downsampled to 5.000 timesteps, is shown in Fig. 5.7. It can correctly detect the current robot room 97.5% of the time and the previously visited room 97.8 % of the time (this result is consistent if different randomly generated reservoirs are considered). Examples of the successful trajectories generated by the HRC system after training are shown in Fig. 5.8. The robot starts in one of the rooms in a position indicated by a circle and navigates to the goal room (given as input) with the end position represented by a small cross. The trajectory is drawn such that its color incrementally changes from green to blue, representing the progress of the navigation. In



(a) E4 - current room

(b) E4 - previous room

(c) E5 - current room

(d) E5 - previous room

Figure 5.7: Performance results of the localization module in environments E4 and E5. Predicted locations are represented by black points whereas solid grey lines are the true robot location. Black crosses represent mistakes.

Fig. 5.8(c), it is shown that the trained system can easily recover from a kidnapping event. The robot started at room 1 and aimed at room 3 as a goal. After reaching room 3, its goal changed back to room 1, but few timesteps later it was kidnapped to room 2. It is possible to see that although it was displaced to another room, the robot was able to drive successfully to its destination (goal room 1), showing that it correctly recognizes the room the robot is located at, which in turn, affects the operation mode of the sensory-motor coupling of the navigation reservoir. This result is consistent across multiple trials and experiments. In 63 routes that were evaluated, the HRC controller has been able to successfully drive the robot to the destination room in all cases without any collision.

The localization performance on test data for environment E5 is shown in Fig. 5.7(c). The system can detect the current and previously visited room 96.33% and 93.63% of the time, respectively. An example of a successful trajectory in environment E5 is shown in Fig. 5.9(a). The robot, driven by the HRC controller, starts at

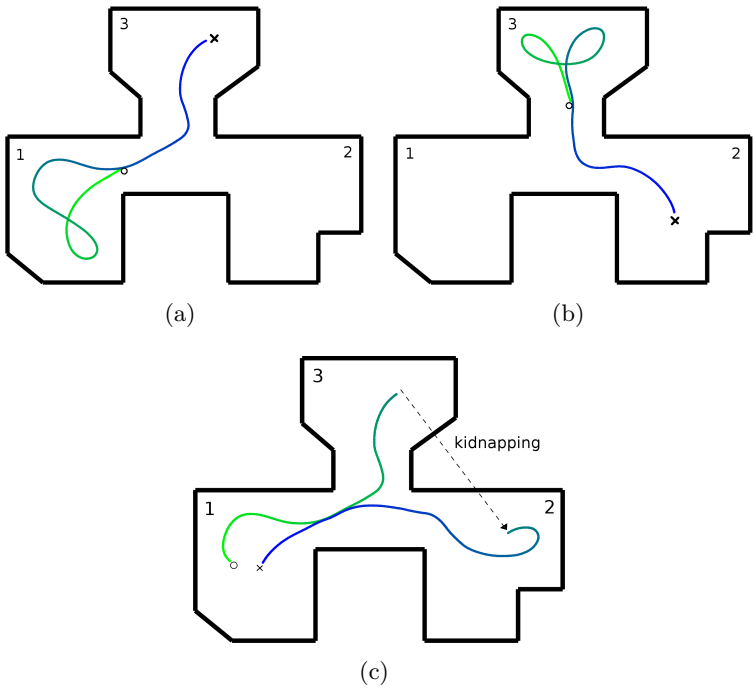
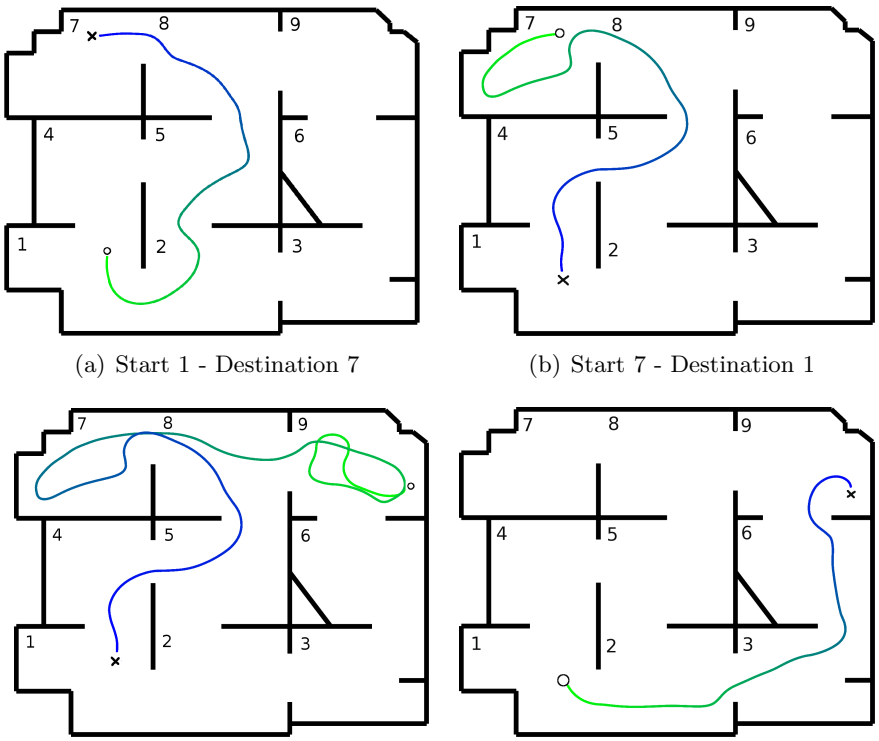


Figure 5.8: Trajectories for robot driven by the HRC controller in environment E4. (a) Robot starts at room 1 and goes to room 3. (b) Robot starts at room 3 and goes to room 2. Starting and ending positions are marked with a circle and a cross, respectively. (c) The robot drives from room 1 to goal room 3. In room 3, its goal changes back to room 1, but it is kidnapped to room 2 after few timesteps. The trajectory shows that it recovered nicely from the kidnapping once it drove directly back to room 1.

room 1 and reaches room 7 successfully. In 15 out of 23 runs, the robot was able to follow the optimal (shortest) path to its goal. In all 23 runs it was able to complete the task. Task completion means that the robot reaches the goal location, being acceptable that during navigation it takes a wrong decision and then goes back to the correct path (see Fig. 5.9(b) for an example). This also shows that the HRC controller is robust to noise and unpredictable situations since it is able to reach the destination even though the robot looses itself for a moment when it mistakenly enters a room outside the shortest path. A summary of the experimental results is given in Table 5.3.

It is important to observe that most of the errors of the localiza-



(a) Start 1 - Destination 7

(b) Start 7 - Destination 1

(c) Start 9 - Destination 1

(d) Start 1 - Destination 3 and Start 3 - Destination 9

Figure 5.9: Trajectories for robot driven by the HRC controller in environment E5. Starting and ending positions are marked with a circle and a cross, respectively. (a) Starting at room 1 and going to target room 7 via rooms ($2 \rightarrow 5 \rightarrow 8$) (optimal path). (b) Starting at room 7 and going to target room 1 via rooms ($8 \rightarrow 5 \rightarrow 4$) (optimal path). (c) Starting at room 9 and going to target room 1 via rooms ($8 \rightarrow 7 \rightarrow 8 \rightarrow 5 \rightarrow 4$) (task completion). (d) Two routes: Starting at room 1 and going to target room 3 via room 2; and starting at room 3 and going to target room 9 via room 6.

Table 5.3: Performance Results in Number of Trajectories

	Shortest Path	Task completion
Environment E4	63 out of 63 (100%)	100%
Environment E5	15 out of 23 (65%).	100%

tion module are made at the transitions between one room and the following one. These errors represent a temporary confusion, which is better than a permanent mistake. Although navigation does not

start in intermediate rooms in environment E5 during testing, it is expected that the robot can reach any goal location regardless of its initial position as long as the same sub-route appears during training. Generalization has been tested to the extent of the kidnapping event. Future work should confirm that the trained system can avoid dynamic unseen obstacles during testing while reaching the desired goal locations. This generalization capability is expected to work with the proposed architecture once it has been shown that reservoir architectures can learn and generalize obstacle avoidance behaviors (Chapter 3).

5.7 Conclusion

This chapter has proposed an hierarchical HRC architecture, composed of *localization* and *navigation* RC networks or modules. The localization module constructs an internal model of the environment from a set of 8 low-accuracy distance sensors which is used by the navigation module to steer the robot through the environment. These networks are trained under a supervised learning framework, which collects data using a supervisor controller (a program or a human supervisor) which generates a series of examples of trajectories from a start room to a destination room. The resulting trained HRC controller is able to perform goal-directed navigation in simple and complex simulated unstructured environments, without the use of odometry and based only on the learned implicit *model* of the environment.

In this chapter, more complex model-based sensory-motor coupling is achieved by combining the results of two previous chapters. The RC network responsible for localizing the robot in its environment (the focus of Chapter 4) can *guide the selection of navigation behaviors* or attractors implemented by another RC network (the focus of Chapter 3). Therefore, by making the navigation module *aware of the context of the robot*, i.e., where the robot is and where it came from (apart from the goal given as external input), it is able to generate a sequence of navigation attractors, defined by a reactive sensory-motor coupling, which lead the robot to the destination room. Actually, additional experiments (not shown in this chapter)

have proved that the prediction of the previously visited room is very important for successful trajectories towards the destination room.

The navigation module integrates different sources of information such as from the distance sensors, the output of the localization module and the goal location, being able to produce behaviors which contain reactive (obstacle avoidance) and deliberative (decision making) components.

The different modules in the HRC architecture operate at distinct timescales for agile processing of low-level sensory-motor behaviors as well as for slow processing of higher-level concepts such as environmental rooms. This is achieved by setting the leak rate of each reservoir to values appropriate for the respective task or skill which it implements, such as localization (*slow* timescale given by a low leak rate) and navigation (*fast* timescale given by a high leak rate) (relevant works such as (Yamashita and Tani, 2008) also elaborate on a hierarchy of slow and fast networks for humanoid robot skill learning).

The current method requires no special landmarks to be placed in the environment and works with inexpensive small mobile robots which have few noisy infra-red distance sensors. Although the environment rooms appear to be different in shape from each other, it has been show that the localization performance is not deteriorated if the environment has multiple symmetric rooms (Antonelo et al., 2008a) (Chapter 4). In this case, eventual confusions, i.e., misclassification of rooms, are not permanent since a short time interval is enough for recovering or correct room recognition.

The supervised learning process requires the labeling of environmental rooms in the training data. This labeling becomes an undesirable work if it is not automatically done. What if an RC-based network is able to autonomously construct implicit abstract environment representations based on raw data from few noisy distance sensors of a small mobile robot? This is the focus of Chapter 7. Although that chapter does not show experiments on goal-directed navigation, it extensively analyses the properties of the architecture in simulated and real environments.

Goal-directed navigation could be learned in a more realistic way under the reinforcement learning framework. The challenge is

to deal with sensor-aliasing problems, non-Markovian environments, continuous state spaces, continuous action space and so on. A first attempt is given in Chapter A which uses primitive discrete behaviors to learn context-dependent navigation attractors in simple environments.

The localization module of the architecture predicts locations based on input from distance sensory readings. Next chapter will focus on generative modeling of environments and robot behaviors, and it shows that it is possible to predict the distance sensors given the robot location as input. Not only that, also robot behaviors can be autonomously simulated, without feedback from the environment.

6 Generative Modeling of Environment-Robot Dynamics

Previously, an implicit spatial representation of an environment has been learned through the projection of the sensory input into the high dimensional space of the reservoir. These RC networks learned *forward models* of the robot by predicting the position from the sensory signals. In this chapter, it is shown that these implicit maps can be made *explicit*, by running the RC network in reverse: predict the local sensory signals given the location of the robot as input (*inverse model*). Furthermore, a Generative RC architecture is proposed, which trains all sensory and localization nodes. With this, it is possible to autonomously (without interference from the environment) generate a predictive sequence of sensory signals and locations, which can be observed at the generative nodes and used to predict the future outcome of robot behaviors. This closed-loop simulation provides a way to internally go through navigation experiences, without actually experiencing with the physical robot body itself (such as *dreaming*), which characterizes a typical planning-like ability.

6.1 Introduction

In order for a mobile robot to become aware of its environment and the possible places which it can be reached by autonomous navigation, it is necessary to create implicit environment models and planning mechanisms which can be efficiently simulated before real execution of the plan is accomplished.

Chapter 4 has shown that an RC network can perform robot localization in complex, dynamic environments using a limited number of distance sensors with low-accuracy, and with a restricted amount of computational power. These RC networks learned *forward models* of the robot which predict the robot position given the sensory signals as input (Fig. 6.1).

In this chapter, it its shown that the same type of RC networks can be used in reverse by predicting local sensory signals from the location of the robot as input, characterizing the learning of an *inverse model* (Fig. 6.1). This new learning scheme enables the **generation of explicit maps of the robot environment** which were implicitly learned in previous chapters.

By simultaneously learning forward and inverse models using a single *generative RC network* which predicts both the sensory signals as well as the position of the robot, it is possible to generate future scenarios of a specific navigation behavior by autonomous simulation of the network in closed loop mode, i.e., without interference of the environment. In this way, navigation trajectories can be internally simulated without actually having to experience them in the actual environment, in a way similar to *dreaming*, which ultimately leads to potential applications in **path planning**. This internal simulation scheme is also speculated in a cognitive point of view in Germund and Hesslow (2002) to happen in mammals, where chains of behavior and perception could be internally simulated (inside the brain).

In addition, a higher-level navigation system can be built with a scheme where the decision module considers these long-term predictions as relevant input. This can be accomplished by modeling several simple reactive behaviors and internally generating long-term expectations for each one in order to switch to the most appropriate behavior at some point in time.

Learning an inverse model with a generative RC network is very useful, for instance, when sensors get broken or occluded. If these **faulty sensors** can be detected, then their values can be predicted based on the learned inverse model which can predict sensory signals given the position of the robot (and possibly other non-faulty sensors). In this way, fault-tolerant localization systems are made possible through the use of self-predicting sensory nodes in the generative RC architecture.

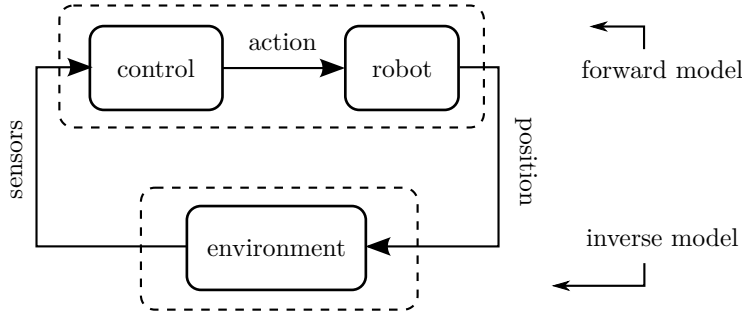


Figure 6.1: Forward and inverse models of a mobile robot. Previous chapters which use RC networks for localization have learned the forward model of the robot, which estimates positions from sensory signals. In this chapter, it is shown that generative RC networks can learn the inverse model as well, that is, they can estimate the sensory signals from the position of the robot.

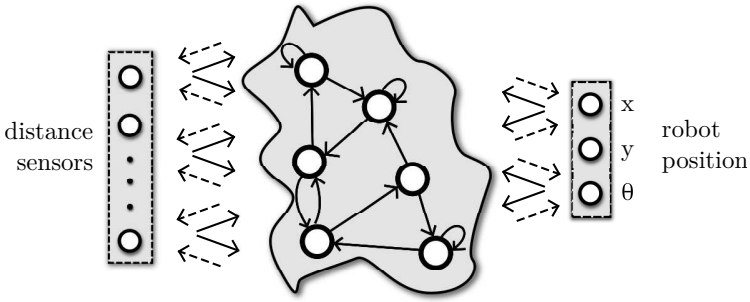
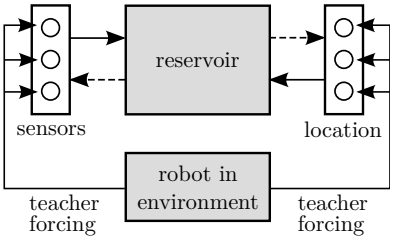


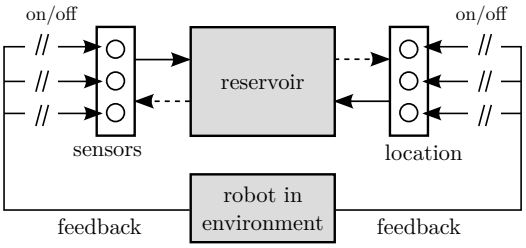
Figure 6.2: Generative Reservoir Computing (GRC) architecture. Input and output nodes become *generative output nodes* in the GRC architecture. All these generative nodes can be trained via teacher forcing, and afterwards, used to predict either the original input signal (i.e., distance sensors) or environment features such as the robot position.

6.2 Generative Reservoir Computing

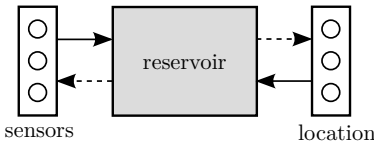
Still under a supervised learning framework, *Generative Reservoir Computing* (GRC) enables the creation of a long-term memory by implementing closed loops between output nodes and reservoir. All nodes of the GRC architecture are called here **generative output nodes** (see Fig. 6.2). These nodes are trained via *teacher forcing*, i.e., information from the environment is written into these output units as if they were input units (Fig. 6.3(a)) (see Section 2.5 for more details). Teacher forcing is typically accomplished during



(a) **Training** the GRC architecture by teacher forcing



(b) **Simulation** of the GRC architecture with optional environment feedback



(c) **Internal simulation** of the GRC architecture (free-run mode)

Figure 6.3: Operation modes of the GRC architecture. (a) Training the GRC architecture occurs by teacher forcing all units, i.e., by stimulating the reservoir with the external feedback (sensors, locations) from the environment as if they were input nodes. (b) Simulation of the GRC architecture can be done by feeding back the sensors from the robot and predicting the location as output, feeding back the location of the robot and expect the distance sensors as output, or optionally feedback any sub-set of the nodes and predict the missing information at the other nodes. (c) Internal simulation of the GRC architecture (or free-run mode), by cutting all feedback from the environment, and letting the network autonomously generate perceptions and path plans, by using only predictions of the network as the next input to the reservoir.

training, but it can be used after training as well. The non-teacher forced outputs operate in **free-run mode**, which means that they self-predict their next values which, in turn, are fed back to the reservoir. Thus, after training, units in free-run mode generate a sequence of predictions in a closed loop, which disregards interference from the environment.

Fig. 6.3(b) shows the GRC architecture after training, where some output nodes can be connected in a closed loop so that they feedback their self-predictions, whereas other nodes may still be used as inputs via teacher forcing. If the location is predicted and the sensors are teacher forced, the GRC architecture models the *forward*

model in Fig. 6.1. If the location is teacher forced and the sensors are predicted, the GRC architecture models the *inverse model* in Fig. 6.1.

Fig. 6.3(c) shows the GRC architecture when all loops are closed for free-run mode operation, so that no external inputs stimulate the reservoir via teacher forcing but only the self-predictions generated by the reservoir itself at the output nodes. By training this architecture with a sequence of samples containing distance sensory readings and the actual robot position (generated by an arbitrary controller), after training it can autonomously regenerate the sequence of sensory readings of the robot and its respective position by letting all nodes self-predict themselves in a closed loop (free-run mode). This ultimately leads to modeling of reactive behaviors which can be used for path planning, as it will be described later in this chapter.

6.3 Generative Modeling of Maps

In the location detection task (Chapter 4), an RC network is used to predict the robot location given distance sensors as input. It thus constructs an implicit map of the environment that is used for localization. Here the reverse problem is considered, that is, given the robot position as input, the reservoir has to predict the expected sensory input, by turning off the environment feedback only for the sensory nodes in Fig. 6.3(b). In this way, by driving the robot in the environment and recording its pose sequence, the RC network can be trained for generating a map of the environment (or the local sensory perceptions of it). The learned implicit map can thus be made explicit, which is useful for evaluating the localization properties of RC-based systems. The learning task considered here is analogous to the modeling of the *inverse model* of the environment-robot dynamics (Fig. 6.1).

It will be shown that not only the continuous-valued robot's pose (x, y coordinates and heading) can be used for generating maps, but also higher-level *locations* represented by a binary vector (see Section 4.1.1 for more details on different types of locations).

The two environments in Fig. 4.7 from Chapter 4, long T-maze (S4) and complex room environment with dynamic obstacles (S5),

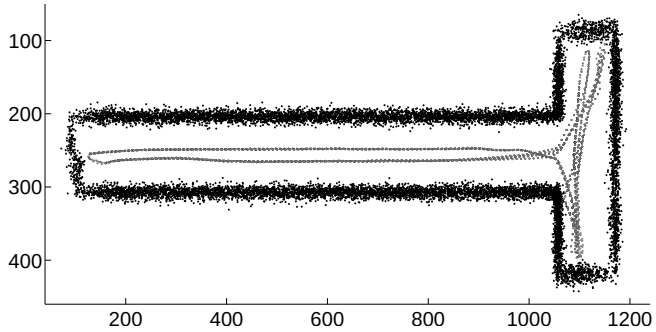
Table 6.1: Parameter configuration for map generation

Model (Env.)	SINAR (S4)	SINAR (S5)
Number of input channels	$n_i = 3$	$n_i = 3$
Input connection fraction	$c_i^r = 0.4$	$c_i^r = 0.5$
Input scaling	$\upsilon_i^r = 0.4$	$\upsilon_i^r = 0.6$
Input downsampling	$d_t = 30$	$d_t = 50$
Input to output connections	yes	yes
No bias		
Reservoir size	$n_r = 400$	$n_r = 800$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 1$	$\alpha = 1$
Number of output channels	$n_o = 17$	$n_o = 17$
Output connection fraction	$c_o^r = 0.4$	$c_o^r = 0.3$
Output feedback scaling	$\upsilon_o^r = 0.4$	$\upsilon_o^r = 0.025$

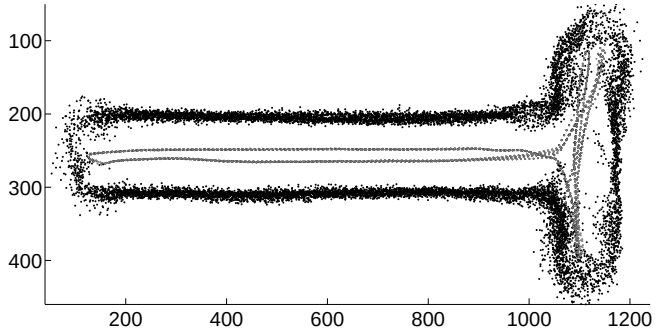
are used for the experiments in this chapter. Both environments contain stochastic aspects which make the robot controller generate diverse random trajectories. A description is given in Section 4.4.3.1.

6.3.1 Settings

The parameter configuration for map generation is given in Table 6.1. It basically uses the same parameters as for the pose estimation task, shown in Section 4.4.4.1 from Chapter 4 for both environments S4 and S5, except for the following changes. The inputs to the reservoir are the normalised robot coordinates (x, y) and heading (θ) whereas the readout output layer, which has feedback connections to the reservoir, is composed of $n_o = 17$ output nodes, corresponding to the distance sensors of the robot. Additionally, the downsampling used for the map learning experiments is made through **decimation**, i.e. taking 1 sample every n timesteps (so, leaving out n-1 samples), so that corners are better represented (otherwise the map is inconsistent). The influence of adding color information as input to the network will also be analyzed.



(a) Noisy Map as *seen* by the robot' sensors



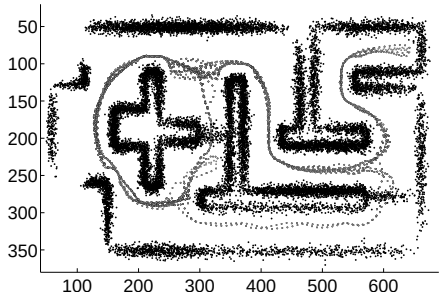
(b) Map generated from **robot's pose**

Figure 6.4: Original and predicted maps for long T-maze (S4) from Fig. 4.7. Black points represent the sensory readings whereas gray points are the robot position. (a) the real noisy map as seen by the robot. (b) the map generated by the RC network after training given the robot coordinates and heading as input.

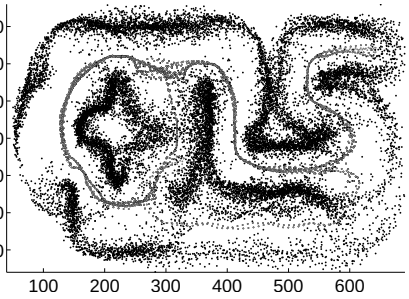
6.3.2 Results

The maps are built by moving the robot according to the pre-recorded test data from the simulator and plotting the predictions of the distance sensor readings from the robot's local coordinate system (see Fig. 6.4). The maps for the test data are generated by running the RC network for 1,300 and 2,000 timesteps for environments S4 and S5, respectively.

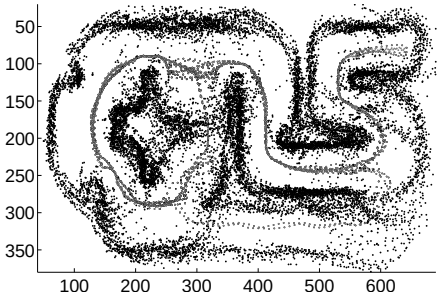
For the long T-maze environment, the generated map (Fig. 6.4(b)) is very similar to the real map. Good performance is also achieved for the more complex maze environment (Fig. 6.5) considering either the robot coordinates or the locations as input (the maps are very similar).



(a) Noisy Map as *seen* by the robot' sensors



(b) Map generated from **robot's pose**



(c) Map generated from **discrete locations**

Figure 6.5: Real and predicted maps for complex room (S5) from Fig. 4.7. Black points represent the sensory readings whereas gray points are the robot trajectory. (a): the real noisy map. (b): the map generated by the RC network trained with the map in (a) (the input to the reservoir is the robot's pose x, y, θ). (c) the generated map considering the more abstract location as input.

The map generation is performed by stimulating the reservoir with the normalized continuous-valued robot coordinates (and heading) or the binary vector of locations; and collecting the expected sensory readings.

The robot trajectory follows a dynamics which is probably used by the reservoir for map generation. In order to find out how the trajectory dynamics is related to reservoir performance, a new experiment was setup with environment S4 by placing four additional attractive objects at the longest corridor. The dashed arrows in Fig. 6.7(a) indicate the positions of attractive objects. Gaussian noise is also added to motor output. Thus, the robot trajectory is diversified with these new changes.

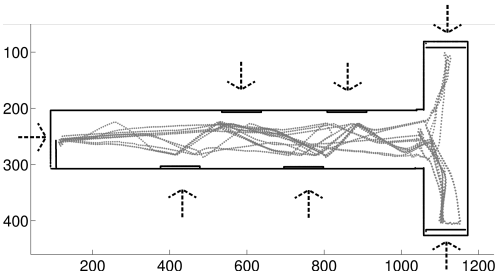
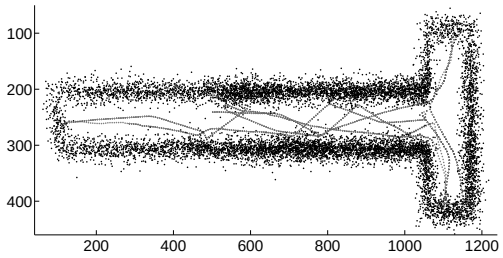


Figure 6.6: Original (non-noisy) map with more irregular and random robot trajectories for the long T-maze (S4). The dashed arrows indicate the position of attractive objects, with only one randomly chosen to be visible at a time (until the robot captures it, when it disappears and a new one is made visible); see Section 4.4.3.1 for more details.

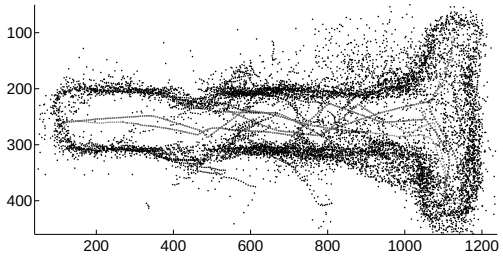
The map in Fig. 6.7(a) is built from 2,800 samples. The noisy version of the map used for training (Fig. 6.7(b)) and corresponding predictions (Fig. 6.7(c)(d)) are constructed from 750 samples. If the reservoir only considers distance sensors as input (ignoring color sensory data), the irregular trajectories in the long corridor cause the miss-prediction in the form of displaced wall segments (Fig. 6.7(c)). By considering the additional color information which is self-predicted by the reservoir during map generation (see architecture in Fig. 6.8), the generated map is significantly improved and no shifted walls are present (Fig. 6.7(d)). Therefore, it is necessary that the network models extra information from the environment (such as the color sensor stream) in order to cope with complex stochastic trajectories in the map generation task. Note that the process of including new information into the prediction model is straightforward, requiring just the addition of more generative sensory outputs which are teacher-forced during training (but are self-predicted after training).

6.4 Planning with GRC architecture: Path Generation

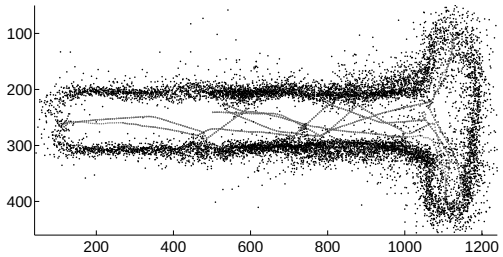
So far, RC networks have been used for both position detection and map generation with the same reservoir configuration, but not with the same network. This section presents an interesting setting in which a single GRC architecture operates completely in free-run



(a) Noisy Map as *seen* by the robot' sensors



(b) Map generation (**disregarding color sensors**)



(c) Map generation (**predicting also color sensors**)

Figure 6.7: Map generation using the trajectories from Fig. 6.6. (b): the map, as seen by the noisy sensors of the robot, used for training (noise added from $N(0, 9)d.u.$). (c): the map generated by GRC network **excluding color sensory data** (teacher forced output: robot's pose; output in free-run mode: distance sensors). (d): the map generated by GRC network **considering color sensory data** using architecture from Fig. 6.8 (teacher forced output: robot's pose; output in free-run mode: distance and color sensors).

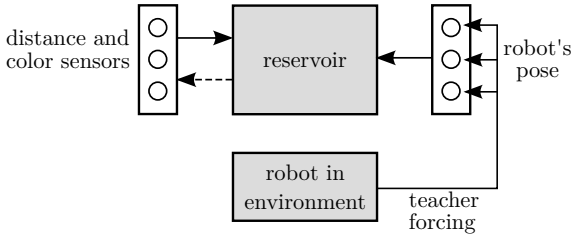


Figure 6.8: Simulation of the GRC architecture with distance and color sensors units in free-run mode, whereas the robot's pose x, y, θ is *teacher-forced* to generative output units.

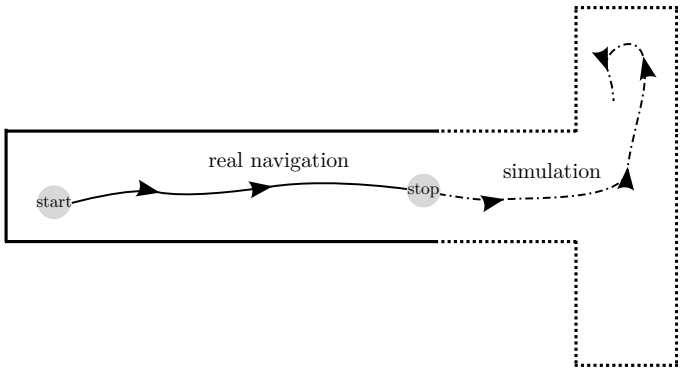


Figure 6.9: From navigation in real environment to free-run simulation mode. Before autonomous simulation is initiated, the robot starts navigating in the actual environment until it stops and begins internal simulation of path plans. The reservoir is stimulated since the start of the navigation and, by the time the robot stops, the reservoir continues in free-run mode (without feedback from the environment) considering the last state it was before starting in this new operation mode. Thus, internal simulation starts from a *situated* point in the physical environment.

mode after training, as shown in Fig. 6.3(c), simultaneously acquiring localization and map generation capabilities. With this setting, the network predicts a sequence of sensory perceptions of the environment and the respective position of the robot, which can be used to predict the future outcome of a robot behavior and its environment. Fig. 6.9 presents a high-level example of how this can be used to internally predict the trajectory of the robot after stopping it in the real environment at some point.

Table 6.2: Parameter configuration for path generation

Model (Env.)	SINAR (S4)
Data downsampling	$d_t = 30$
No bias	
Reservoir size	$n_r = 700$
Reservoir connection fraction	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 1$
Number of generative output channels	$n_o = 20$
Output connection fraction	$c_o^r = 0.4$
Output feedback scaling	$v_o^r = 0.5$

6.4.1 Settings

The RC network is configured as in Table 6.2 for the experiments in this section. The readout layer has $n_o = 20$ generative output nodes (17 for distance sensors plus 3 for normalised robot coordinates and heading). The noise in the state update equation during the generation of the matrix $\mathbf{X}$ with the harvested reservoir states is given by the Gaussian distribution $N(0, 0.0001)$ (see Sections 2.5 and 2.6 for more details).

6.4.2 Results

The experiments are accomplished in the long T-maze environment S4. The robot navigates continuously through this environment using the controller from Antonelo et al. (2006), capturing targets which are positioned at the 3 corners of the maze, with only one visible at a time (see Fig. 4.7), while its sensors and position are recorded. The resulting dataset (the same as previous section) contains 3,600 timesteps after resampling. The RC network is trained with the first 2,400 timesteps. Then, the initial state of the reservoir is set to the state it was at the final timestep 2,400 of the training dataset.

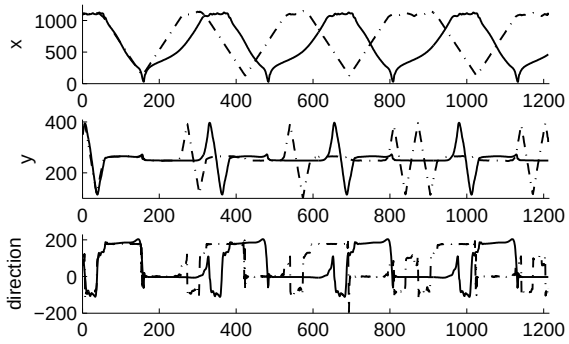
After that, the reservoir operates in free-run self-predicting mode, where every network output is fed back to the reservoir. In other words, the robot navigates physically for 2,400 timesteps in the environment and then it autonomously simulate its trajectory in the en-

vironment (including the local sensory perceptions) using the GRC network. Fig. 6.9 gives an example of how this procedure is executed. As the output is not teacher-forced, the reservoir is free to develop its own dynamics. The results are shown in Fig. 6.10. The first plot shows the predicted robot position whereas the second one corresponds to the autonomously generated map. As it can be observed, the predicted robot trajectory follows a cyclic dynamics as if the robot was capturing each target in a predefined sequence. Probably this sequence is the most relevant in the training dataset. The generated map shows that the long corridor is well rebuilt while the corners are more difficult to reconstruct due to the fast turning behavior at these locations.

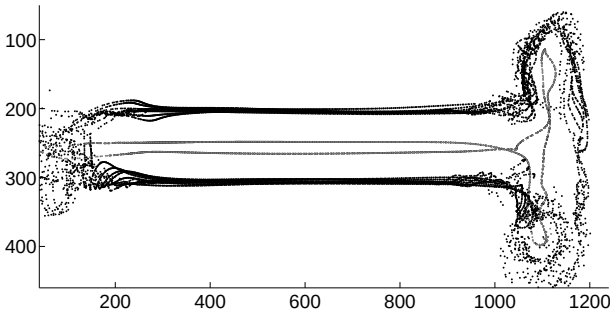
This chapter demonstrates that a single RC system is able to both model the environment and generate trajectories and environment perceptions in a free-run fashion. These capabilities are processed by a single recurrent neural network without any rule-based mechanism or higher-order technique. On the other hand, this neural network model could be used by some more abstract decision-based system, for instance: for making future predictions on the robot's path and environment if a certain behaviour is chosen from some point on (in this case a behavior would be an input channel to the GRC network).

Although the term might seem quite colloquial this can be understood as if the robot was *dreaming* about its environment and its associated reactive behaviour (see Germund and Hesslow (2002) for a well described, cognitive-based hypothesis on this subject which applies for mammals in general). Furthermore, this can be achieved with a system that is biologically plausible (Yamazaki and Tanaka, 2007), which allows us to speculate on whether simpler animals could possess this capability for short-term predictions (or *anticipation*) of their actions in their environment[1].

[1] According to Rosen (1985), all living organisms are anticipatory systems, which means that they can predict the future state of the environment with predictive models disregarding the existence of a mental process.



(a) Predicted robot path



(b) Generated map

Figure 6.10: Path generation in free-run mode, without environment feedback. The RC network predicts both robot position and the sensory readings (there is no teacher forcing during prediction). (a): the predicted robot position. (b): the corresponding generated map (black points are the sensory readings and gray points are the predicted robot trajectory).

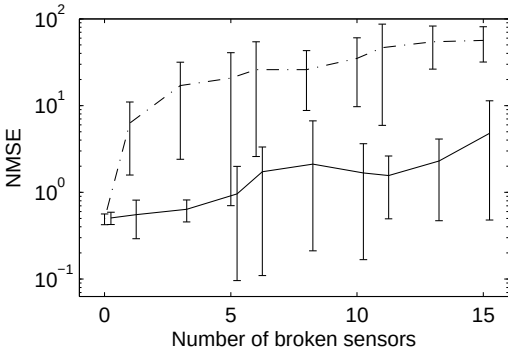


Figure 6.11: Comparison of RC and GRC architectures in a robot localization task in environment S5 when sensors get broken. The dashed line is the average error using the standard RC network whereas the solid line represents the average error using the generative GRC architecture which predicts the value of broken sensors.

6.5 Fault-tolerant Localization

This section presents a way of improving the robustness of RC-based location detectors (from Chapter 4) against faulty or broken sensors. In order to achieve a fault-tolerant localization system, a GRC network must be trained to predict not only the locations, but also the sensory readings. Assuming that faulty sensors can be detected, then the sensory nodes which present malfunction will be replaced by self-predicted values.

In Fig. 6.11, a normal RC-based pose estimator (dashed line) is compared to the GRC-based pose estimator which predicts the values of the faulty sensors (solid line). Note that the vertical axis of the plot is in a logarithmic scale. The task of pose estimation (x, y, θ) is accomplished using environment S5 and the same parameter configuration as described in Section 4.4.4.1. Each experiment is executed 20 times, where the broken sensors are randomly chosen for each test dataset. This is done by setting their outputs to zero (in the case of the standard RC-based estimator). As the number of broken sensors increases, the localization performance of the usual RC network deteriorates notably but gracefully. On the other hand, the generative setting of the GRC network which predicts the values of broken sensors can cope very well with the missing sensory

information.

6.6 Conclusion

The *Generative Reservoir Computing* (GRC) architecture can predict any sensory, motor, or high-level concept by training all *generative output nodes* using teacher-forcing (stimulating the reservoir with the desired signal through the output nodes). During testing, the output nodes can serve as input units by teacher forcing the signal from the environment while other nodes can feedback their self-predictions into the reservoir. Using this approach, locations can be predicted from sensory readings (forward model) as well as sensory readings can be predicted from locations (inverse model). A GRC-based localization system can even tolerate to a great extent missing information from faulty sensors, when the prediction of these sensors are used instead.

When all output nodes of the GRC architecture feedback their self-predictions into the reservoir, thus disregarding signals from the environment, it is possible to explicitly see that the network embedded the navigation behavior of the robot during training. The internal simulation of a sequence of local environment perceptions and robot positions reveal a reactive navigation attractor which can be used to predict future scenarios of a specific robot behavior in the current environment.

Chapters 3 and 5 show that multiple navigation attractors can be embedded in the same RC network. External input channels or even hidden units from another (localization) network can be used to change the behavior of the RC navigation network. By converting this network into a generative GRC network, it is possible to generate path plans not only of one behavior but of multiple different behaviors. In a environment where rewards are given for each time the robot reaches a room in a predefined sequence, it is possible to train a GRC network with an extra output unit representing the reward, so that, during testing, it is able to perform path planning (by internal simulation of different reactive behaviors) and find the rewarded route.

A problem observed during map generation is that fast turning

behavior is not well captured by the RC network. The local environment perceptions close to these areas of fast turning are not well enough predicted (note the malformed corners of the long T-maze, mainly with respect to the experiment of path planning). This may be caused by resampling artifacts, or even by the lack of diversity of timescales in the reservoir. Future work could be done to improve this by either using a reservoir with multiple leak rates or two disconnected pools of neurons in the reservoirs each one with a different leak rate (for enabling the generation of uncorrelated signals of different frequencies).

In this and all previous chapters, the training of RC networks has been done in a supervised way. Next chapter covers the unsupervised learning of RC networks using a fast and efficient method for training the linear readout output layer, called Slow Feature Analysis. With this, sensor-based environment representations emerge in a self-organized way which can be used to localize small robots in unknown, unstructured environments.

7 Unsupervised Learning for Robot Localization

In order to achieve a higher degree of autonomy in the learning process of RC-based navigation systems which use implicit learned models of the environment for goal-directed navigation, this chapter tackles the unsupervised learning of these implicit models. Thus, unlike previous chapters, labeling training data with the robot location is **not** required in this chapter. To this end, a hierarchical architecture is proposed which has the reservoir as a temporal non-linear kernel in the first layer, and an upper layer which is trained using the technique called *Slow Feature Analysis* (SFA). SFA is able to learn invariant or slowly varying output signals from a given input signal in an unsupervised way. It is shown experimentally in this chapter that the proposed RC-SFA architecture is empowered with an unique combination of short-term memory and non-linear transformations which overcomes the *hidden state problem* present in robot navigation tasks. In addition, experiments with simulated and real robots indicate that spatial activations generated by the trained network show similarities to the activations of CA1 hippocampal cells of rats (a specific group of neurons in the hippocampus).

7.1 Introduction

Most reservoir computing models use supervised learning schemes to train the readout output layer. In this case, linear regression is the standard technique used for output training (Jaeger and Haas, 2004). However, biological systems probably learn a great number

of tasks in an unsupervised way.

Slow Feature Analysis (SFA) (Wiskott and Sejnowski, 2002) is an unsupervised learning method based on the concept of slowness. It extracts invariant or slowly-varying representations of a high-dimensional input signal, and has been shown to be able to model properties of complex cells from the primary visual cortex V1 (Berkes and Wiskott, 2005). As SFA only learns linear mappings from an input signal, a non-linear quadratic expansion of the original input signal is typically accomplished before SFA is applied.

In this chapter, a hierarchical architecture is proposed, where the first layer comprises an (usually sparsely connected) non-linear recurrent network with fixed weights (the reservoir), and the second layer consists of SFA units. The short-term memory of the reservoir and its non-linear projection in conjunction with such an unsupervised learning technique yields a model which possesses advantages from both theoretical models: the inherent spatiotemporal processing capabilities of the reservoir as well as the slowly-varying hidden signal extraction of the SFA model. I call the proposed architecture the RC-SFA model.

The slowness extraction mechanism present in SFA allows that high-level concepts, such as the position or orientation of a subject inside a room, which are slowly varying in time, be generated from low-level fast-varying stimuli like vision. In the same way, the location of a mobile robot inside an environment can be predicted from vision, but also from distance sensors, for instance.

This chapter shows that, using the proposed RC-SFA model, small mobile robots are able to learn to self-localize in simulated and real environments in an unsupervised way based only on few noisy infra-red distance sensors and no proprioceptive inputs (e.g. disregarding odometry signals or path integration). The network receives the raw low-dimensional input signal, which is projected to a high-dimensional, dynamic reservoir space, used by the SFA layer to generate spatially dependent representations of the environment. The activation of SFA units is spatially non-localized (they exhibit low-place selectivity), that is, after training, they exhibit high activity for multiple locations of an environment. A second step is necessary for producing units which are only active for no more than one location, exhibiting high-place selectivity. In Franzius et al. (2007a)

an additional post-processing step using Independent Component Analysis (ICA) is applied for learning sparse representations from SFA units. Similarly, the proposed RC-SFA model uses ICA in the third layer as the second step for generating localized representations of a robot environment. The complete architecture is shown in Fig. 7.1.

Experiments with rats in open fields have shown the existence of hippocampal place cells. These cells form an implicit spatial representation of an animal's environment, firing whenever the rodent is located at a particular location (O'Keefe and Dostrovsky, 1971; O'Keefe, 1976; Moser et al., 2008), which defines the place field of the cell. The biological inspiration to RC-based localization systems and further information on hippocampal place cells and other types of spatial encoding cells are presented in Section 1.2.4.1.

Although the majority of the existing place cell models are not dependent on the animal's direction of movement, there are few experiments showing that place cells exhibit movement-related firing patterns such that the environment configuration and the animal's behavior can impose a directional structure in the firing of place cells (Eichenbaum et al., 1999). In Brunel and Trullier (1998), it is proposed that place cells are intrinsically directional and that invariance to direction is achieved through generalization. In Frank et al. (2000), using a constrained environment for rats such as W tracks, it is shown that hippocampal CA1 cells and entorhinal cortex (EC) cells code for spatial information on a way dependent on the rat's path or behavior. It assumes that, if the hippocampus and EC are related to path planning over extended trajectories, these structures should reflect where the animal intends to go or where it has come from. Similarly, the proposed RC-SFA architecture in this work encodes positional information on a path-dependent way, where the SFA layer and ICA layer exhibit an activation pattern comparable to that of EC cells and hippocampal CA1 cells found in Frank et al. (2000), respectively.

The current work is inspired by the fact that whiskers of rodents can provide relevant information about the environment and shape of objects (Solomon and Hartmann, 2006). In the same way, the experiments in this chapter are based on a small mobile robot which perceives the environment through a limited number of short-range

distance sensors. This work assumes that the low-dimensional input, such as whiskers for rats or distance sensors for robots, can provide interesting information from the environment.

7.2 Related works

It has been shown in Franzius et al. (2007a) that a hierarchy of SFA layers with increasing receptive fields at upper layers and a top ICA layer can be trained to code for either the rat's position or the rat's head direction depending on the movement pattern of a simulated rat. Their model is based on the high-dimensional input from a camera which simulates the 320^o field of view of the rat. Simple environments such as linear tracks or rectangular arenas with distinct textures set for each wall make it possible to infer the rat's position from a single image. The similarities between their model and the one proposed in this paper refer to the layers which learn by SFA and ICA. The main differences are that the proposed model uses a dynamic reservoir at the first layer, which projects a low-dimensional input into a high-dimensional non-linear space and which proved to be essential for learning spatial representations with such a small number of distance sensors. Moreover, the proposed model copes with sensor aliasing, where multiple environmental states map to the same perceptual sensory input. This means that it is not sufficient to consider only the current time step to determine the robot location, but the history of the input stream - a property which the reservoir naturally has.

In Wyss et al. (2006), a cortical hierarchy of layers is proposed which learn by optimizing an objective function that takes into account temporal stability and temporal decorrelation between units. All units are leaky integrators providing them with a local memory trace. Their method is similar to Slow Feature Analysis in the sense that it maximizes temporal stability or slowness of the output signal. However, their learning method is iterative and, as so, prone to convergence to a local optimum (unlike SFA). Their experiments are made with a mobile robot driving randomly in a rectangular environment with predefined cues. The continuous stream of a 16x16 pixel image feeds the architecture which, after learning, shows prop-

Table 7.1: New variables used in this chapter

$\mathbf{y}_{\text{SFA}}$	output of SFA units
$\mathbf{y}_{\text{ICA}}$	output of ICA units
$\mathbf{W}_{\text{SFA}}$	Connection matrix for SFA units
$\mathbf{W}_{\text{ICA}}$	Connection matrix for ICA units
n_{SFA}	number of SFA units
n_{ICA}	number of ICA units

erties of hippocampal place cells.

The SFA algorithm has also been used as the training method of a linear readout output layer of reservoirs of spiking neurons in Klampfl and Maass (2009). They show that these linear readouts can learn to discriminate isolated spoken digits in an unsupervised way.

7.3 Methods

7.3.1 RC-SFA architecture

The proposed RC-SFA architecture is composed of a hierarchical network of nodes where the lower layer is a reservoir and the upper layers are composed of SFA and ICA units, respectively (Fig. 7.1). This hierarchical network learns in an unsupervised way. The function of the reservoir is to map the inputs to a high-dimensional dynamic space. Because of its recurrent connections, the reservoir states contain *echoes* of the past inputs, providing a short-term memory to the model. The SFA layer receives signals from the input nodes $\mathbf{u}[n]$ and from the reservoir nodes $\mathbf{x}[n]$. This layer generates invariant or slowly varying signals (Wiskott and Sejnowski, 2002) which are instantaneous functions of input from previous layers (see Section 7.3.2). The upper-most layer is composed of ICA units which generate a sparse and local representation, as in sparse coding[1], of the slowly varying SFA features. The following sections focus on these upper layers, while Table 7.1 presents a list of new variables related to the RC-SFA architecture.

[1]**Sparse coding** is *the representation of items by the strong activation of a relatively small set of neurons* (Foldiak and Endres, 2008).

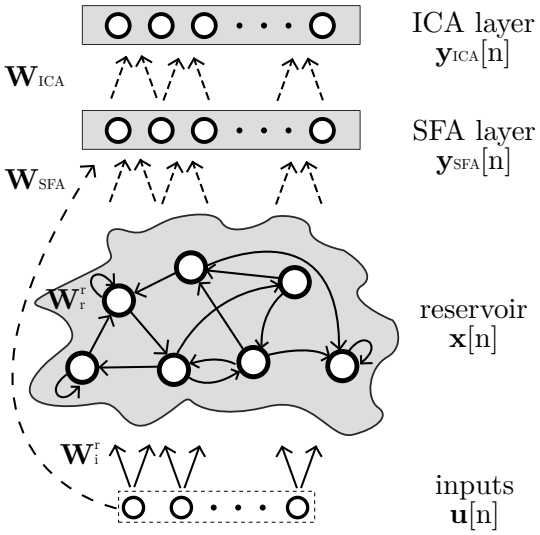


Figure 7.1: RC-SFA architecture. The reservoir is a recurrent network where the inputs are mapped to a high-dimensional non-linear space. The resulting reservoir trajectory, generated by input stimulation, is used for training the SFA layer, which extracts instantaneous slowly-varying signals from the reservoir after learning. The subsequent ICA layer implements sparse coding on the SFA outputs, extracting independent components from the SFA activation. Training is unsupervised and takes place for the dashed connection lines in the figure. Solid lines represent fixed randomly generated weights.

7.3.2 Slow Feature Analysis

Most sensory input signals vary at a fast timescale, even though the environment properties may be slowly varying. This is because sensors provide low-level representations of environment features and are prone to fast signal variations, e.g., a human moving inside an office produces fast visual input variation while his/her position in the building changes slowly. Slow Feature Analysis (SFA) (Wiskott and Sejnowski, 2002) is an algorithm which finds output functions $g_i(\mathbf{x}(t))$ which maximize temporal slowness, given a high-dimensional input signal $\mathbf{x}(t)$. It extracts functions which try to provide a higher level representation of the environment, assuming that they vary in a slower timescale when compared to the raw input. The SFA output is an instantaneous function of the input so that it depends only on the current state, differently from a filter which depends on previous inputs[2].

In mathematical terms (Wiskott and Sejnowski, 2002), the SFA model tries to find output signals $y_i = g_i(\mathbf{x}(t))$ such that:

$$\Delta(y_i) := \langle \dot{y}_i^2 \rangle_t \qquad \text{is minimal} \tag{7.1}$$

under the constraints

$$\langle y_i \rangle_t = 0 \qquad \text{(zero mean)} \tag{7.2}$$

$$\langle y_i^2 \rangle_t = 1 \qquad \text{(unit variance)} \tag{7.3}$$

$$\forall j < i, \langle y_i y_j \rangle_t = 0 \qquad \text{(decorrelation and order)} \tag{7.4}$$

where $\langle . \rangle_t$ and $\dot{y}$ denote temporal averaging and the time derivative of y, respectively.

Learning

The first step of the learning process is normalizing the input signal $\mathbf{x}(t)$ to have zero mean and unit variance.

The common step of non-linear expansion of the input signal is not used in this work, but it is replaced by the **non-linear reservoir** at the first layer of the RC-SFA architecture. It can be shown that

[2] Although the SFA output does not itself implement a filter, the reservoir in the first layer of the RC-SFA architecture acts as a non-linear filter.

SFA learning corresponds to solve a generalized eigenvalue problem (Wiskott and Sejnowski, 2002):

$$\mathbf{AW} = \mathbf{BW}\Lambda, \tag{7.5}$$

where $\mathbf{A} := \langle \dot{\mathbf{x}}\dot{\mathbf{x}}^T \rangle_t$ and $\mathbf{B} := \langle \mathbf{x}\mathbf{x}^T \rangle_t$.

The eigenvectors $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{n_{\text{SFA}}}$ corresponding to the ordered generalized eigenvalues $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{n_{\text{SFA}}}$ solve the learning task, satisfying (7.2-7.4) and minimizing (7.1) (see Wiskott and Sejnowski, 2002, for more details). This algorithm is guaranteed to find the global optimum. Learning and inference is very fast, as there are efficient methods for solving the generalized eigenvalue problem. The decorrelated SFA outputs extract instantaneous slowly-varying signals, which is different from low-pass filtering of the inputs.

Although the eigenvalue problem for solving SFA is biologically unrealistic, biologically plausible implementations of SFA exist (Hashimoto, 2003).

Architecture

The SFA layer (Fig. 7.1) is denoted by $\mathbf{y}_{\text{SFA}}(t)$:

$$\mathbf{y}_{\text{SFA}}[n] = \mathbf{W}_{\text{SFA}}\mathbf{x}_{\text{SFA}}[n], \tag{7.6}$$

where $\mathbf{x}_{\text{SFA}}[n]$ is a normalized input vector at time step n consisting of a concatenation of input $\mathbf{u}[n]$ and reservoir states $\mathbf{x}[n]$. Note that the states $\mathbf{x}[n]$ are generated by stimulating the reservoir with the input signal $\mathbf{u}[n]$ for $n = 1, 2, ..., n_{\text{s}}$ by using the state update equation (2.9) with leaky-integrator units, where n_{s} is the number of samples.

The weight matrix $\mathbf{W}_{\text{SFA}}$ is a $n_{\text{SFA}} \times (n_i + n_r)$ matrix corresponding to the eigenvectors found by solving (7.5).

7.3.3 Independent Component Analysis

Independent Component Analysis (ICA) is a method used for sparse coding of input data as well as for *blind source separation* (Hyvärinen and Oja, 2000). The ICA model assumes that a linear mixture of signals $x_1, x_2 ... x_n$ can be used for finding the n indepen-

dent components or latent variables $s_1, s_2...s_n$. The observed values $\mathbf{x}(t) = [x_1(t), x_2(t)...x_n(t)]$ can be written as:

$$\mathbf{x}(t) = \mathbf{A}\mathbf{s}(t), \tag{7.7}$$

where $\mathbf{A}$ is the mixing matrix and $\mathbf{s}(t) = [s_1(t), s_2(t)...s_n(t)]$ is the vector of independent components (both $\mathbf{A}$ and $\mathbf{s}(t)$ are assumed to be unknown). The vector $\mathbf{s}(t)$ can be generated after estimating matrix $\mathbf{A}$:

$$\mathbf{s}(t) = \mathbf{W}\mathbf{x}(t), \tag{7.8}$$

where $\mathbf{W} = \mathbf{A}^{-1}$.

The basic assumption for ICA is that the components s_i are statistically independent. In Hyvärinen and Oja (2000), it is also assumed that the independent components have non-Gaussian distributions. In their work, an optimal estimator for nongaussianity is given by **negentropy**:

$$J(\mathbf{y}) = H(\mathbf{y}_{\text{gauss}}) - H(\mathbf{y}), \tag{7.9}$$

where $\mathbf{y}$ is a random vector; $\mathbf{y}_{\text{gauss}}$ is a Gaussian variable with the same covariance as $\mathbf{y}$; and H is the information-theoretic measure of **differential entropy**:

$$H(\mathbf{y}) = -\int f(\mathbf{y}) \log f(\mathbf{y}) \mathrm{d}\mathbf{y}, \tag{7.10}$$

where $f(\mathbf{y})$ is the density of $\mathbf{y}$ (note that the Gaussian variable has the *largest entropy of all random variables with the same variance*). They approximate the negentropy measure J by:

$$J(y) \propto (E\{G(y)\} - E\{G(\nu)\})^2 \tag{7.11}$$

where G is a nonquadratic function; ν is a standardized Gaussian variable (zero-mean and unit variance).

In their work, ICA is formulated as **minimization of mutual information**, which is an information-theoretic measure of the independence of random variables. The mutual information I among

m scalar variables is given by:

$$I(y_1, y_2, \ldots, y_m) = \sum_{i=1}^{m} H(y_i) - H(\mathbf{y}), \tag{7.12}$$

and is related to negentropy by:

$$I(y_1, y_2, \ldots, y_m) = C - \sum_{i=1}^{m} J(y_i), \tag{7.13}$$

where C is a constant, assuming that the components y_i are *uncorrelated* and of unit variance. Thus, from (7.13), minimizing the mutual information is equivalent to finding the directions (or subspace projections) in $\mathbf{W}$ which *maximize the negentropy* (Hyvärinen and Oja, 2000).

Learning

In this chapter the matrix $\mathbf{W}$ is found with the FastICA algorithm (a detailed derivation, based on maximization of negentropy, is given in Hyvärinen and Oja (2000)). Before using ICA, the observed vector $\mathbf{x}(t)$ is preprocessed by centering (zero-mean) and whitening (decorrelation and unit variance, i.e., make $E\{\mathbf{x}(t)\mathbf{x}(t)^T\} = \mathbf{I}$) (Hyvärinen and Oja, 2000). FastICA uses a fixed-point iteration scheme for finding the maximum of the non-Gaussianity of $\mathbf{w}_i\mathbf{x}(t)$ (where $\mathbf{w}_i$ is a weight vector of one neuron). The basic form of the FastICA algorithm (for one unit) is described next:

1. Initialize $\mathbf{w}_i$ randomly
2. Let $\mathbf{w}_i^+ = E\{\mathbf{x}g(\mathbf{w}_i^T\mathbf{x})\} - E\{g'(\mathbf{w}_i^T\mathbf{x})\mathbf{w}_i\}$
3. Let $\mathbf{w}_i = \mathbf{w}_i^+ / \|\mathbf{w}_i^+\|$
4. Do steps 2 and 3 until convergence,

where g is the derivative of the nonquadratic function G (in this work, $g(u) = u^3$ for simulation experiments and $g(u) = u\exp(-u^2/2)$ for the experiments in real environments). Convergence means that vectors $\mathbf{w}_i^+$ and $\mathbf{w}_i$ point in the same direction. The next units $\mathbf{w}_i$ in $\mathbf{W}$ are found one by one such that the outputs $\mathbf{w}_i^T\mathbf{x}$ are decorrelated

This algorithm finds a direction for $\mathbf{w}_i$ such that the projection $\mathbf{w}_i^T\mathbf{x}$ maximizes non-Gaussianity. This means that the independent components will mostly be clustered, concentrated on specific values

(in contrast with the more random values of Gaussian variables).

Whereas the FastICA algorithm may seem biologically unrealistic, an alternate biologically plausible implementation of ICA can be achieved through non-linear Hebbian learning (Hyvärinen and Oja, 1998).

Architecture

The equation for the ICA layer is (by redefining variables):

$$\mathbf{y}_{\mathrm{ICA}}[n] = \mathbf{W}_{\mathrm{ICA}}\mathbf{y}_{\mathrm{SFA}}[n], \tag{7.14}$$

where: $\mathbf{y}_{\mathrm{SFA}}[n]$ is the input vector at time step n (the observed values); $\mathbf{W}_{\mathrm{ICA}}$ is the mixing matrix ($n_{\mathrm{ICA}} \times N_{\mathrm{SFA}}$); and $\mathbf{y}_{\mathrm{ICA}}[n]$ is the output of the ICA layer (the independent components).

After training, the ICA units are ordered by kurtosis

$$\mathrm{kurt}(y_{\mathrm{ICA}}) = \frac{E\{y_{\mathrm{ICA}}^4\}}{(E\{y_{\mathrm{ICA}}^2\})^2} - 3 \tag{7.15}$$

such that the first unit has the most kurtosis. The above expression simplifies to $E\{y_{\mathrm{ICA}}^4\} - 3$ once we assumed y_{ICA} is of unit variance.

7.3.4 Place cell reconstruction

It is common to use population vector coding for interpreting activation from hippocampal place cells (Zhang et al., 1998). However, Bayesian methods have shown superior performance for reconstructing the position of freely moving rats and were shown to be biologically plausible (Zhang et al., 1998). In this chapter, a probabilistic method based on Zhang et al. (1998) is used to estimate the robot position from the activation of ICA units. The reconstruction is based on the conditional probability

$$P(\mathbf{x_r}|\mathbf{y}_{\mathrm{ICA}}) = \frac{P(\mathbf{y}_{\mathrm{ICA}}|\mathbf{x_r})P(\mathbf{x_r})}{P(\mathbf{y}_{\mathrm{ICA}})}. \tag{7.16}$$

The prior $P(\mathbf{x_r})$ is the probability of the robot being at position $\mathbf{x_r} = (x, y)$ and can be estimated from with the true recorded position during robot navigation. $P(\mathbf{y}_{\mathrm{ICA}})$ is a normalization term which does not need to be explicitly known. $P(\mathbf{y}_{\mathrm{ICA}}|\mathbf{x_r})$ is the probabil-

ity of the activation $\mathbf{y}_{\text{ICA}}$ given that the robot is at location $\mathbf{x_r}$. As ICA units $\mathbf{y}_{\text{ICA}}$ are statistically conditionally independent, the conditional probability factorizes and can be computed as:

$$P(\mathbf{y}_{\text{ICA}}|\mathbf{x_r}) = \prod_{i=1}^{n_{\text{ICA}}} P(y^i{}_{\text{ICA}}|\mathbf{x_r}), \tag{7.17}$$

where $P(y^i{}_{\text{ICA}}|\mathbf{x_r})$ is the probability of ICA unit i given that the robot is at position $\mathbf{x_r}$, which can be estimated by computing the histogram of the data $P(y^i{}_{\text{ICA}}|\mathbf{x_r}) = \langle y^i{}_{\text{ICA}}\rangle_{\mathbf{x_r}}/\mu$, where $\langle y^i{}_{\text{ICA}}\rangle_{\mathbf{x_r}}$ is the mean activation of ICA unit i in position $\mathbf{x_r}$ and μ is a normalization factor.

The reconstructed position is given by:

$$\hat{\mathbf{x}}_{\mathbf{r}} = \arg\max_{\mathbf{x_r}} P(\mathbf{x_r}|\mathbf{y}_{\text{ICA}}), \tag{7.18}$$

which is the most probable position given the ICA layer activation.

7.4 Experiments with Simulated and Real Robots

This section shows that, using the RC-SFA architecture, the capability of self-localization of small mobile robots can emerge in a self-organized way. Experiments are accomplished in a simulated environment using the SINAR robot model, which has 17 short-range noisy distance sensors (see Section 3.2.1 or the Appendix B.1 for more details), as well as in real-world environments using the e-puck robot (see Section 4.2.1 or Appendix B.2), which has 8 infrared distance sensors. The real-world experiment is more difficult for two main reasons: an increased stochasticity of the robot controller (described in Section 4.2.1.3) and a limited number of sensors. Based only on the local information from few noisy sensors, the RC-SFA architecture can autonomously learn an internal representation of the environment which allows for spatial coding and self-localization.

7.4.1 Environments

For the SINAR robot model, the experiments are conducted using environment S6 (Fig. 7.2), a big maze with 64 predefined locations

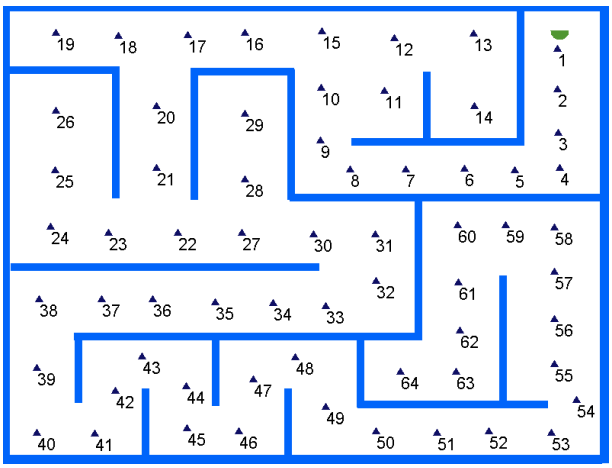
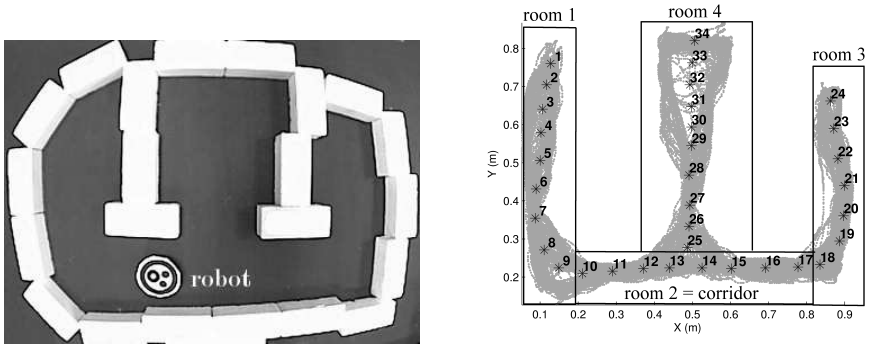


Figure 7.2: SINAR environment S6, as in Fig. 4.8, tagged with 64 labels displayed by small triangles. Dimensions are given in *distance units* (d.u.). *This figure is reproduced here a second time to facilitate the comprehension of the following experimental results.*

spread evenly around the environment (represented by small labeled triangles).

For data generation, the simulated robot navigates in the environment for 180.000 timesteps while its distance sensor measurements are recorded. It takes approximately 13.000 timesteps for the robot to visit most of the predefined locations with the SINAR controller described in Section 3.2.1, which basically makes the robot explore the whole environment. Additional investigations are also performed with a modified version of the environment containing 11 obstacles which randomly move around the environment, representing an extra source of noise which also changes the robot behavior and trajectory in the environment.

The environment of the e-puck robot has 3 rooms and a connecting corridor (see environment E3 in Fig. 4.10). The robot navigates in this environment according to the controller described in Section 4.2.1.3. So, it can stay navigating in one room for a random time interval, eventually making ellipsoid trajectories or leaving the room towards the corridor (see Fig. 4.10(b)). The randomness of the robot movement is determined by τ (see Section 4.2.1.3), which is the probability of changing the movement direction at each second. ICA units would learn to code for locations by the performing ex-



(a) Real environment for e-puck (E3) (b) Robot trajectory

Figure 7.3: Environment and trajectory of the real e-puck robot, as in Fig. 4.10. (a) Environment (120 cm × 90 cm) composed of three rooms and one corridor. The position of the robot is tracked with a camera mounted above the environment for analysis purposes. (b) Trajectory in gray generated by the robot controller in environment E3 for 60.000 timesteps (or 3.3 hours), with labeled asterisks representing delimited locations. *This figure is reproduced here a second time to facilitate the comprehension of the following experimental results.*

periments with different settings $\tau = 0, \tau = 0.02, \tau = 0.03$, although the more random the movement, the more difficult the place cell learning. The results shown in this chapter consider the most random behavior, that is, $\tau = 0.03$, which practically means that there is a probability of circa 60% for inverting the direction of movement while the robot is navigating inside one of the rooms. The total number of recorded samples is 192.000, which means approximately 11 hours of robot navigation.

7.4.2 Settings

For model optimization, multiple grid search experiments are performed over subsets of the model parameters, using the place cell reconstruction method from Section 7.3.4 for position estimation. Each experiment during the optimization process is executed 10 times, with each run considering a randomly generated reservoir.

First, the recorded input signal $\mathbf{u}[n]$ is used to generate the reservoir states $\mathbf{x}[n], n = 1, 2, ..., n_s$ using (2.9). Then, the training of the RC-SFA architecture is accomplished in 2 steps. First, the SFA layer is trained by solving (7.5) where the inputs are the reservoir states

Table 7.2: Parameter configuration (values in bold were found by parameter search)

Model (Environment)	SINAR (S6)	e-puck (E3)
Number of input channels	$n_i = 17$	$n_i = 8$
Input connection fraction	$c_i^r = 0.3$	$c_i^r = 0.3$
Input scaling	$\upsilon_i^r = 0.9$	$\upsilon_i^r = 2$
Input downsampling	$d_t = \mathbf{50}$	$d_t = \mathbf{1}$
No bias		
Reservoir size	$n_r = \mathbf{300}$	$n_r = \mathbf{600}$
Reservoir connection fraction	$c_r^r = 1$	$c_r^r = 1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.99$	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.99$
Leak rate	$\alpha = \mathbf{0.4}$	$\alpha = \mathbf{0.1}$
Number of SFA units	$n_{\mathrm{SFA}} = \mathbf{128}$	$n_{\mathrm{SFA}} = \mathbf{128}$
Number of ICA units	$n_{\mathrm{ICA}} = \mathbf{128}$	$n_{\mathrm{ICA}} = \mathbf{128}$

and distance sensors (as in (7.6)). After $\mathbf{W}_{\mathrm{SFA}}$ is found, the output of SFA units $\mathbf{y}_{\mathrm{SFA}}[n], n = 1, 2, ..., n_s$ is generated using (7.6). The second step corresponds to training the upper ICA layer by applying the FastICA algorithm from Section 7.3.3 where the inputs for this layer are the output of the SFA units. The output signals $\mathbf{y}_{\mathrm{SFA}}[n]$ and $\mathbf{y}_{\mathrm{ICA}}[n]$ are upsampled to the original sampling rate of $\mathbf{u}[n]$.

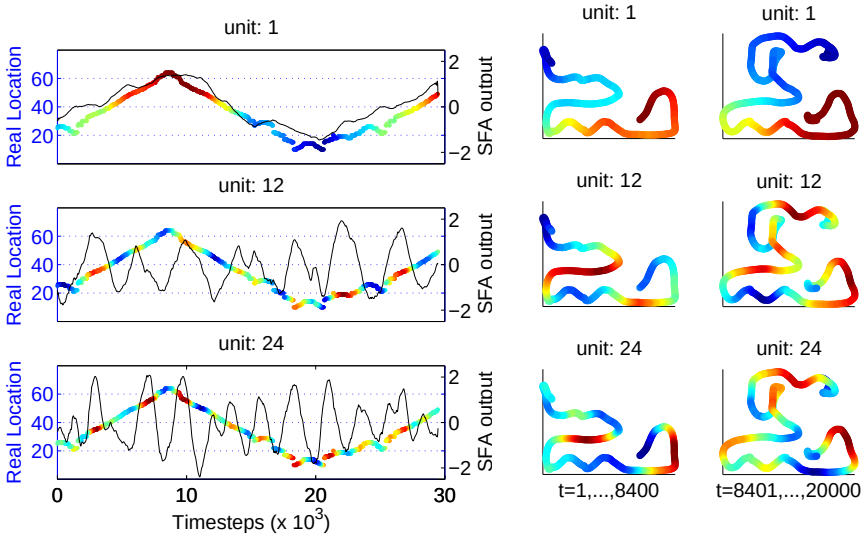
The parameter configuration is given in Table 7.2 for both SINAR and the real e-puck robots, where values in bold denote optimal parameters.

After downsampling the input signal for SINAR, $n_s = 3.600$. For SINAR, the training dataset has $5n_s/6$ samples and the test dataset has $n_s/6$ samples. For the real e-puck, 9/10 of the input signal (172.800 samples) is used as the training dataset and 1/10 (19.200 samples) is used for testing.

7.4.3 Experimental Results

7.4.3.1 SINAR

Fig. 7.4 shows the output of 3 SFA units for a test input signal. The left plot, Fig. 7.4(a), shows the outputs over time whereas the right plots, Fig. 7.4(b), show the response of the neurons as a function of the robot position in the environment. In the left plot, the



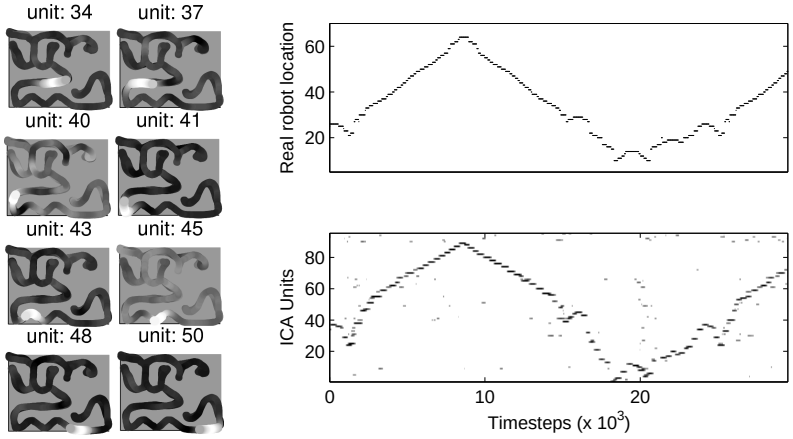
(a) SFA output over time

(b) SFA output as a function of the robot position

Figure 7.4: Response of SFA units 1, 12, and 24 for simulations in Environment S6 on test data. (a) the SFA output over time. For each location (in time) given by the labeled triangles in Fig. 4.8, there is a colored dot where red denotes a peak response, green an intermediate response, and blue a low response. The output is also plotted as a black line. (b) the same SFA output as a function of the robot position for two distinct time invervals $[1, 8400]$ and $[8401, 20000]$.

horizontal axis represents the time, the left vertical axis denotes the real robot location (as given by the labeled triangles in Fig. 4.8), and the right vertical axis denotes the SFA output of the neuron. The colored dots represent the output of the SFA unit (where red denotes a peak response, green an intermediate response, and blue a low response). The SFA output is also shown as a black line in the same plot and as a colored trajectory in the right plot. As SFA units are ordered by slowness, the first SFA unit has the slowest response. It shows a high response for locations 40 to 64 and a low response otherwise. Units 12 and 24 vary much faster, encoding several locations of the environment. In Fig. 7.4(b), each of the units is shown during two different time intervals, $[1, 8400]$ and $[8401, 20000]$. In that way, it is possible to observe that units 12 and 24 encode spatial information in a way which is dependent on the robot path, that is, their activation depends on where the robot has come from in the environment (a characteristic comparable to EC cells of rats in Frank et al., 2000).

The upper ICA layer builds on the SFA layer. During learning, ICA units seek to maximize non-Gaussianity so that their responses become sparse and clustered, and also as independent as possible. This form of sparse coding leads to the unsupervised formation of place cells. Fig. 7.5(a) shows a number of ICA units which code for specific adjacent locations in the environment. The peak response is represented by white dots while lower responses are given gradually in darker colors. As the robot navigates, a sequence of high activity spots (white dots) is observable through these set of ICA units, each one coding for a specific location in the environment. In order to visualize the localized property of place cells more clearly, the output of ICA units are ordered such that they have a spatial relationship. The reference locations (from 1 to 64), shown in environment S6 (Fig. 4.8), are used to automatically order the ICA layer. ICA units which do not respond strongly enough in any situation are removed from the vector. Fig. 7.5(b) shows the real occupancy grid for the robot while it drives in environment S6 and the respective ICA activation map showing the spatially-ordered ICA responses (where $\mathbf{u}[n]$ is a test signal not used during learning). Stronger responses are represented by darker dots in the figure. This activation map is very similar to the real robot occupancy grid showing that the



(a) ICA output (b) ICA Activation map

Figure 7.5: Response of ICA units for simulations in Environment S6 on test data. (a) Response of ICA units as a function of the robot position. White dots denote high activity while darker dots represent lower responses. The results show the localized aspect of place cells or ICA units (the peak response is characteristic of one specific location). (b) The real robot occupancy grid (top) and the respective spatially-ordered ICA activation map (bottom), where black dots denote peak responses and white represent lower responses.

place cells efficiently learned to cover most of the locations in the environment.

The experiments shown here were repeated 15 times with the same datasets, where each time a different random reservoir is created. These distinct experiments have not shown any significant differences in the quality of the learned place cells.

Most ICA units show an activation which is dependent on the direction of the robot's movement. This can be visually confirmed in Fig. 7.6, where a blue surface represents a low activation and a red surface means high activation. It shows that the activation of several ICA units (vertical axis in the figure) is, on average, high only for particular robot directions (horizontal axis in the figure). This is comprehensible since the environment is composed of narrow corridors, which shapes the robot trajectory so that the orientations that the robot may have are restricted by the environment configuration. So, as the robot direction is not, in general, a constantly fast-varying feature, it is also learned by SFA and ICA units.

Using the probabilistic place cell reconstruction method from

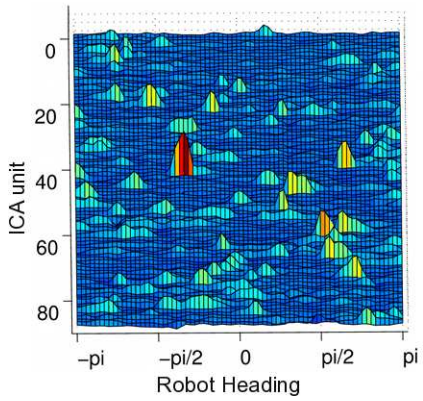
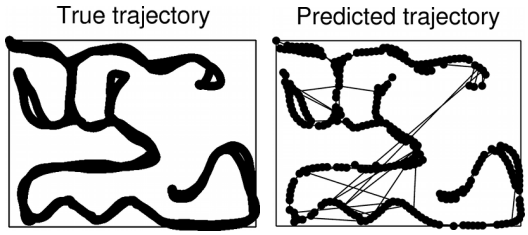


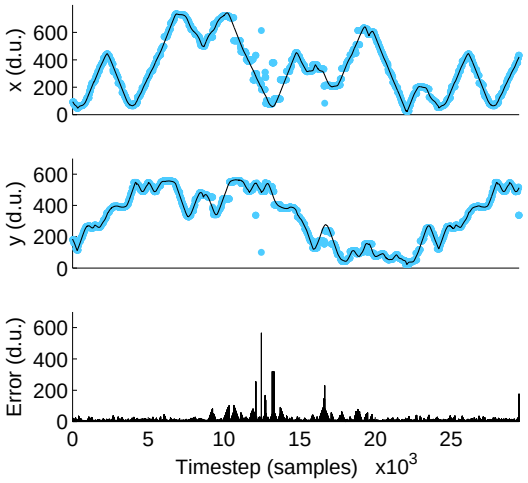
Figure 7.6: The plot shows the mean activation of ICA units as a function of the robot heading, which reveals the dependence of trained ICA units on the direction of movement. Red denotes high response while blue a low response.

Section 7.3.4, the predicted robot position during robot navigation is computed given the activation in the ICA layer. The true and predicted trajectory can be seen in Fig. 7.7(a) showing that the RC-SFA architecture is able to learn an internal spatial representation of the environment. Some jumps in the predicted position can be observed, which also occur with the estimated position computed with signals recorded from hippocampal place cell of rats (Moser et al., 2008). Fig. 7.7(b) shows the same true and predicted positions in terms of the x and y coordinates of the robot in the world frame along with the respective error, given by the Euclidean distance between the true (black line) and the predicted (points in cyan color) positions. The mean test error is 17.2 distance units - see Table 7.3.

Another experiment with environment S6 is performed, in which 11 dynamic obstacles were artificially added to the environment. These obstacles were constantly moving around in a random way, possibly closing passages and forcing the robot to follow another path. That yields more stochasticity in the environment and in the robot behavior. The recorded dataset consists of 200.000 samples. Training and parameter configuration are the same as in the previous experiment. The test error of 108 d.u., shown in Table 7.3, is clearly higher than when the environment is not dynamic. Fig. 7.8 shows the network predictions as points in cyan and the true position as a black curve. Despite this higher error rate, the trained system



(a) Trajectory prediction in S6



(b) Position prediction in S6

Figure 7.7: Prediction of the robot position in environment S6 given the activation in the ICA layer using the place cell reconstruction method on test data. (a) The true robot trajectory as connected black points (left) and the corresponding estimated robot trajectory by the place cell reconstruction method given the activation in the ICA layer (right). (b) The true and predicted robot coordinates given by black curves and points in cyan color (gray for black-and-white prints), respectively. The bottom plot shows the error as the Euclidean distance between true and predicted position.

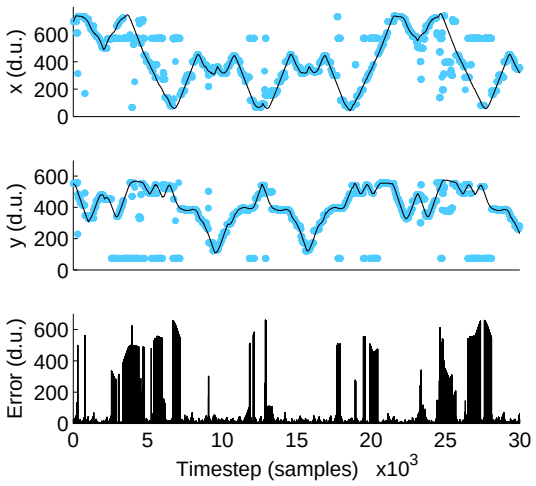


Figure 7.8: The true and predicted robot position in a modified version of environment S6, containing 11 dynamic moving obstacles. Mistakes can be observed when the predicted points in cyan (gray for black-and-white prints) deviate from the black line (also detected by the error plot).

is robust enough to recover from intense environment stochasticity given by dynamic obstacles and a period of miss-predictions without the use of odometry.

7.4.3.2 Real Extended e-puck

The mean activation of 4 SFA units, rescaled to the inverval $[0, 1]$, as a function of the robot position is shown in Fig. 7.9(a). The slowest SFA feature shows a high response in room 1 which gradually decreases as it gets further to room 3. The third slowest feature has a low response in the middle room and a high response otherwise.

Table 7.3: Results using the place cell reconstruction method

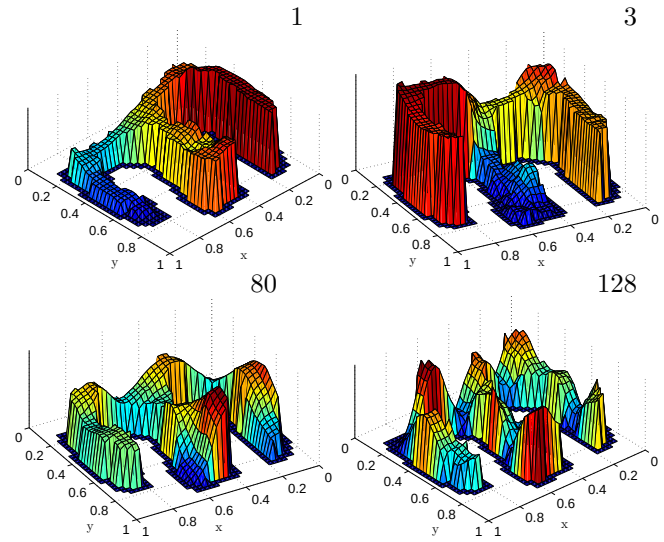
Environment (Robot)	Dimensions	Type	Architecture	Test Error
S6 (SINAR)	800×600 d.u.	Simulation	RC-SFA	17.2 d.u.
S6 dynamic (SINAR)	800×600 d.u.	Simulation	RC-SFA	108 d.u.
E3 (E-puck)	120×90 cm	Real	RC-SFA	11 cm
E3 (E-puck)	120×90 cm	Real	SFA	23 cm

Faster-varying features, like units 80 and 128, show high responses in multiple locations of the environment, characterizing low place selectivity in a way similar to entorhinal cortex cells of rats in Frank et al. (2000).

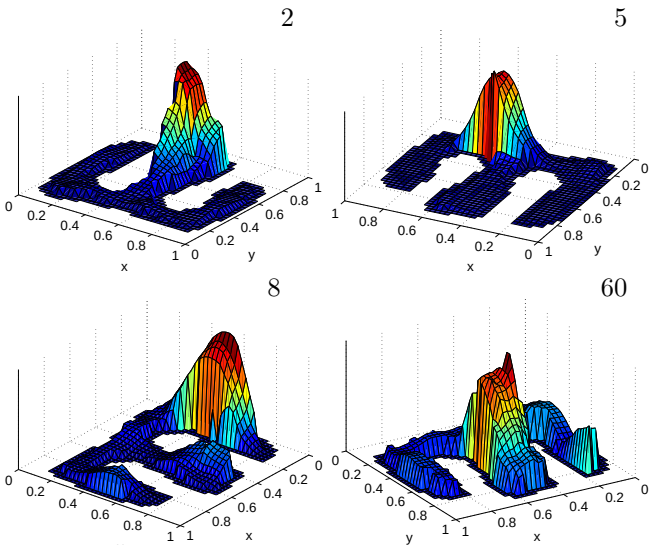
The mean activation of ICA units as a function of the robot position $h_i(\mathbf{x_r})$ is computed by averaging out the response of each ICA unit over a discrete grid of evenly spaced robot positions. Four units' mean activation are shown in Fig. 7.9(b). It is clear that these units learned to code for particular locations in the environment, i.e., the place fields of the cell, presenting a peak response at the center of these locations.

The mean activation does not show whether the unit is invariant to the robot movement direction. To investigate about the directionality aspect of ICA units, Fig. 7.10(a) shows several plots, where each row corresponds to an ICA unit, and each column considers robot positions with specific robot headings. Each plot displays robot positions associated with a heading θ in cyan color, whereas the robot positions plotted in maroon color represent a strong activation of the corresponding ICA unit. For example, unit 5, in the first row, is strongly activated at the right part of the corridor when the robot is heading right ($\theta = 0 \pm \kappa$). The last column of this figure shows the mean activation as a function of the robot heading, clearly showing the direction dependence of these ICA units. Fig. 7.10(b) is another plot which indicates the directionality dependence of ICA units, by showing the mean activations of each ICA unit as a function of the robot heading, where the most representative (with most kurtosis) ICA units are direction-dependent.

By using the probabilistic method described in Section 7.3.4, the capability of trained ICA units is evaluated in terms of robot localization performance. Fig. 7.11 shows the estimated robot position using equation (7.18) as well as the true robot position for 3.000 timesteps. The test error, given by the Euclidean distance between $\mathbf{x_r}$ and $\hat{\mathbf{x}}_\mathbf{r}$ was 0.1188 for these 3.000 timesteps. It can be seen in the figure that the estimated position matches very well with the true robot position, confirming the good localization capability which emerged from the unsupervised learning of the RC-SFA architecture. Furthermore, erratic jumps of the estimated robot position can be seen in this figure, which is actually also observed in the esti-

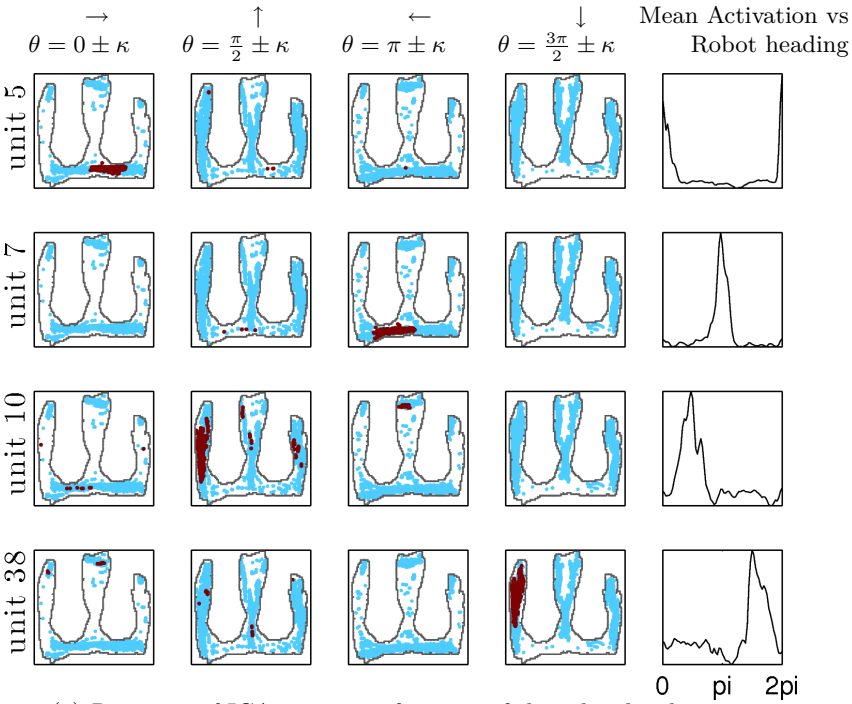


(a) SFA units

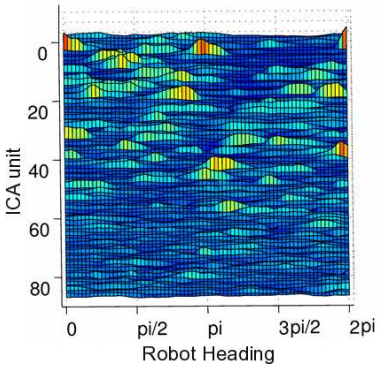


(b) ICA units

Figure 7.9: Activation of SFA and ICA units in environment E3. Red denotes a high response whereas blue denotes a low response. (a) Mean activation of SFA units as a function of the robot position in the environment, rescaled to the interval [0,1]. (b) Mean activation of place cells (ICA units) as a function of the robot position in the environment.

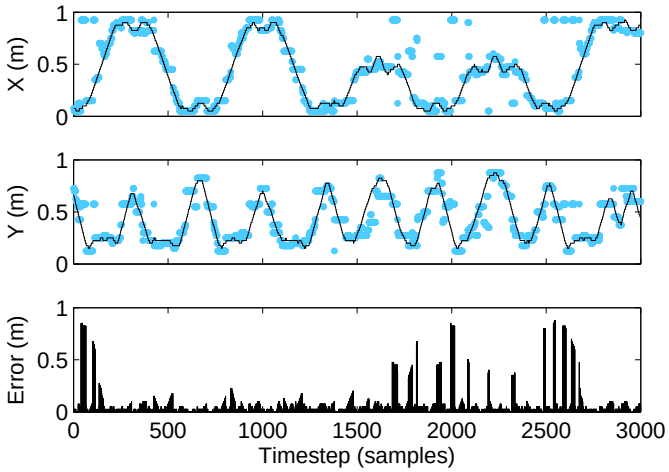


(a) Response of ICA units as a function of the robot heading

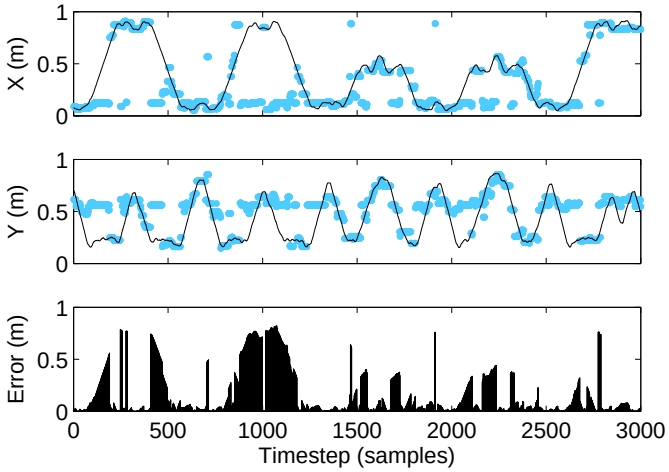


(b) Mean activation of ICA units as a function of the robot heading

Figure 7.10: Results after training with the e-puck robot in environment E3. (a) Directionality dependence of place cell activation for test data. Each row represents a place cell, where points in cyan (lighter) color denote the positions occupied by the robot for given directions θ in the environment and points in maroon (darker) color represent the positions where the place cell responses are higher than a certain fixed threshold. The last column shows the mean activation of a place cell as a function of the robot heading. (b) Mean activation of place cells as a function of the robot heading. The plot shows that the activation of most place cells is dependent on the robot heading. The results are shown for test data. Red denotes a high response whereas blue denotes a low response.



(a) RC-SFA + ICA



(b) SFA + ICA (using a time window and non-linear expansion)

Figure 7.11: Predicted robot position in environment E3 on test data using the Bayesian place cell reconstruction method for 3.000 timesteps of navigation. (a) Results using the RC-SFA architecture. The true and predicted robot coordinates are given by black curves and points in cyan color (gray for black-and-white prints), respectively. The bottom plot shows the error as the Euclidean distance between true and predicted position. (b) Results using an architecture without the reservoir, but with a time window and non-linear expansion on the input signal for the SFA layer.

mated position from the activity recorded from hippocampal place cells of rats (Zhang et al., 1998).

7.4.3.3 Robustness to Noise

The robustness of the proposed architecture was also tested considering different levels of Gaussian noise on the sensor measurements. In Fig. 7.12, the mean and standard deviation of the test error, given by the Euclidean distance between true and predicted positions, are displayed for noise levels ranging from 1% to 50%. The error stays very low even with 15% noise on sensors. From 20% on, sensors become too noisy and do not convey useful information, which causes the error to unexpectedly raise to near a maximum (in a nonlinear way), showing that the architecture is quite robust to noisy sensors. After 20% noise, a generative network could be used to predict sensory signals from the activation of ICA units (see Chapter 8 for future work).

7.4.3.4 The Role of the Reservoir

The dynamics of the reservoir is best fine-tuned by grid searching two parameters: the input scaling of $\mathbf{W}_{\mathrm{i}}^{\mathrm{r}}$ and the spectral radius $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}})$ (Verstraeten and Schrauwen, 2009) (see Section 2.3). Fig. 7.13 shows that the nonlinear reservoir performs better for $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) <= 1$, on average. The reservoir's dynamic nonlinear regime is further tuned by choosing an optimal input scaling. Higher values of the input scaling yield an improvement in performance, as shown in the figure. The optimal combination is an input scaling of 2.5 and $\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$.

Leaky integrator neurons can also enhance performance if the input timescale does not optimally match the reservoir timescale. The leak rate α in (2.9) controls how fast (or slow) reservoir units respond to input stimuli. In Fig. 7.14, it is possible to see that there is an optimum for the leak rate, when it is approximately 0.07. It also shows that the short-term memory in the reservoir is an important characteristic for the learning of the SFA and ICA layers and, therefore, for the performance of the localization capability of the robot.

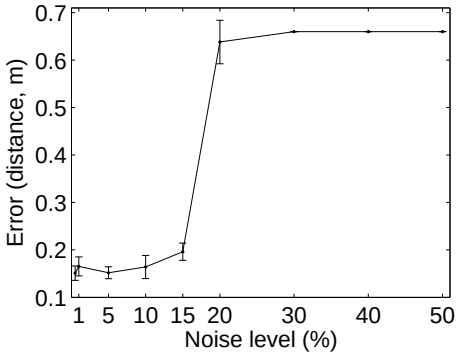


Figure 7.12: Robustness to Gaussian noise. The plot shows the mean and the standard deviation of the localization error on test data from environment E3 considering different noise levels on all 8 robot distance sensors. Training data uses only 0.5% noise on sensors in all experiments. Each experiment is run for 10 times (the plot shows the mean and the standard deviation).

In order to confirm the importance of the reservoir in the proposed RC-SFA architecture, experiments are performed with an architecture which replaces the reservoir by a non-linear expansion on a time-delayed downsampled input signal. The non-linear function expands the input signal in the space of polynomials of degree 2. The model was optimized by grid searching the following parameters: downsampling rate d_t and size of the time window t_w. The best performance in terms of the Euclidean distance between true and predicted robot position, which is 0.21, is attained for $d_t = 32$ and $t_w = 2$.

So, this architecture uses a reduced and smoothed input signal by a downsampling process which converts $n_s = 192.000$ samples into $n_s = 6000$ samples. It also uses a time window of size 2 which effectively produces $n_i = 16$ inputs after the non-linear expansion step. Table 7.3 shows that this model has a test error which is almost double of the error using the RC-SFA architecture. In Fig. 7.11(b), the predicted robot position using this model and its associated error are shown. Although it learned to code for some locations in the environment, the precision is not as good as with the RC-SFA model in Fig. 7.11(a). Intermediate locations are not coded at all: note that the predicted x coordinate often presents big jumps.

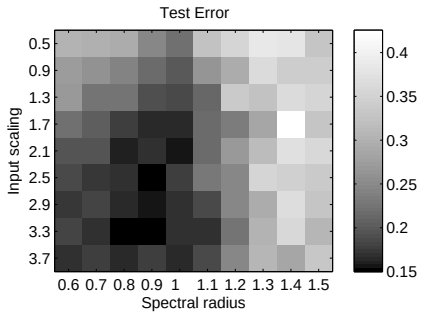


Figure 7.13: Input scaling vs. Spectral radius. The plot shows the mean and the standard deviation of the test localization error in environment E3 for different combinations of input scaling in $\mathbf{W}_{\text{i}}^{\text{r}}$ and the reservoir's spectral radius $\rho(\mathbf{W}_{\text{r}}^{\text{r}})$. Each experiment is executed 5 times with randomly generated reservoir weights. Input scaling of 2.5 and $\rho(\mathbf{W}_{\text{r}}^{\text{r}}) = 0.9$ yields the minimum test error (black surface represents low error while white surface corresponds to higher error).

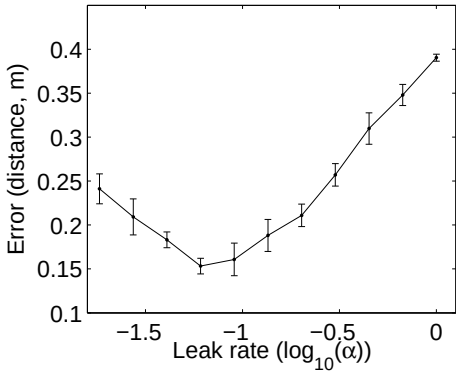


Figure 7.14: Influence of the reservoir's leak rate on performance. The plot shows the mean localization error (and its standard deviation) on test data from environment E3 considering different leak rates α of the reservoir. Each experiment is executed 10 times with randomly generated reservoir weights. $\alpha = 0.07$ yields the lowest test error.

7.5 Discussion

The SFA layer in the proposed model has shown low place selectivity, with similarities to entorhinal cortex (EC) cells of rats in W tracks (Frank et al., 2000), whereas the ICA layer has presented high place selectivity and an activation which is dependent on the robot path, similarly to hippocampal cells of rats in Frank et al. (2000).

The directionality aspect of place cells in the ICA layer is in accordance with other works in the literature which show that environment shape and robot behavior affect the directionality component of place cells (Frank et al., 2000; Eichenbaum et al., 1999; Brunel and Trullier, 1998). Most of these computational models implement path integration using idiothetic input, despite the current lack of knowledge with respect to the mechanisms of path integration in the brain like the integration of self-motion signals with allothetic input (Eichenbaum et al., 1999). On the other hand, in the proposed RC-SFA architecture, the reservoir integrates the allothetic input (distance sensors), forming a trajectory in state space which SFA units use to learn spatial features from a given environment. This computation can be compared with optical flow, in the sense that the reservoir provides a temporal memory of the stimuli stream which can be used for distance estimation, that is, the reservoir is involved in maintaining an estimate of the robot location for a temporary time period in the absence of the distance sensory input.

Learning in SFA is comparable to Principal Component Analysis (PCA) in terms of complexity. Furthermore, while biologically plausible implementations of SFA exist (Hashimoto, 2003), there is experimental evidence showing that the slowness learning principle of SFA is present in the visual cortex (Li and DiCarlo, 2008).

Although the current ICA implementation may seem biologically unrealistic, a more biologically plausible learning scheme for generating place cells at the top ICA layer from the non-localized representation of SFA units can be implemented by competitive learning (Franzius et al., 2007b) or non-linear Hebbian learning (Hyvärinen and Oja, 1998).

7.6 Conclusion

This chapter has proposed a biologically-inspired hierarchical architecture with three layers for learning sensor-based spatial representations of a robot environment in an unsupervised way. The proposed model does not use any idiothetic signals for path integration (or odometry) as most models do (Burgess et al., 2007; Hasselmo, 2008; Arleo et al., 2004; Stroesslin et al., 2005; Milford, 2008), and is the first to rely solely on a limited number of raw distance sensors for unsupervised learning of place cells.

The first layer of the architecture is a reservoir of recurrent nodes, which is used as a form of temporal kernel for projecting low-dimensional inputs (e.g., a small number of distance sensors) to a dynamic high-dimensional space. It integrates the noisy distance sensory input for making it possible to infer the robot position from the history-dependent trajectory of the dynamic reservoir. The second layer learns in an unsupervised way to derive slowly-varying features from the reservoir states and also possibly from the input layer, using the Slow Feature Analysis (SFA) algorithm. These slow features indicate latent signals present in the input signal which vary in a slower timescale, such as the position or the orientation of a robot in its environment. If the position can be inferred from given inputs (e.g., the reservoir state), SFA can extract it based on the slowness concept. The top layer produces independent components which are a linear combination of the SFA features, using Independent Component Analysis (ICA). It learns a sparse coding on the SFA outputs, resulting in units which are activated only for a specific position in the robot environment.

Using a probabilistic place cell reconstruction method (Zhang et al., 1998), the robot position (coordinates in the world's frame) is estimated from the activation in the ICA layer. This estimated position has shown that the ICA layer in the RC-SFA architecture has an activation correlated to the robot position, confirming the powerful capability for sensor-based unsupervised learning of spatial representations. These results are obtained in simulated environments considering a robot model with 17 distance sensors, as well as in real environments using the e-puck robot with 8 infra-red distance sensors.

This chapter represents a step towards enhancing the autonomy of previous architectures which were based on a supervised learning scheme. The RC-SFA architecture enables the **self-organized autonomous learning of implicit spatial representations of a robot environment**, that is, it is not necessary to label training data with the location (or room) of the robot during its navigation as done previously. Moreover, goal-directed navigation systems could significantly benefit by using an unsupervised way of learning internal models with RC-SFA, revealing them as more biologically plausible systems. Future work and extensions are given in Chapter 8.

8 Conclusion and Future work

8.1 Summary

The usual Reservoir Computing (RC) model, called Echo State Network, is composed by a randomly generated sigmoidal recurrent network with fixed weights, the reservoir, and an adaptive linear readout output layer, usually trained by linear regression methods. RC has been considered a major breakthrough in training recurrent networks due to ease of training and good convergence properties of the training method.

This work has shown that RC networks can be trained and used to a multitude of navigation and modeling tasks using small mobile robots with inexpensive sensory apparatus in unknown environments. RC naturally solves **sensor aliasing** problems, common for robot navigation tasks, through learning and computation in the high-dimensional dynamic system space. In this way, the sequence of robot positions given by a reactive robot behavior in an environment can be inferred from the trajectory of the high-dimensional state of the reservoir, for instance. This is because the trajectory of the dynamical reservoir system inherently possesses a short-term, fading memory of previous inputs, thus capable of **disambiguating the sensory input space**. Moreover, all tasks considered in this work are performed using solely a **limited number of noisy distance sensors** as input. Encoder information from wheels for path integration (dead reckoning) are not necessary, besides counting wheel revolutions is probably not biologically plausible.

With RC, robust reactive behaviors can be learned in an **imitation learning** way by providing examples of desired sensory-motor coupling. More interestingly, a single RC network can be trained to model multiple behaviors or **navigation attractors** by changing the operating point of the dynamical reservoir with additional external inputs. These extra inputs create sub-space attractors in the dynamical system space, each one modeling a different behavior, which effectively boosts the memory capacity of RC networks. These **sub-space attractors** can be understood as reactive behaviors which emerge from the coupling existing between the robot controller (the RC network) and the environment. They can be visually inspected by observing the trajectory of the robot in the environment space, or by observing their projection in the high-dimensional reservoir space.

Hierarchical RC networks proposed in this work enabled **complex goal-directed navigation** in an unknown, unstructured environment. The predictions of hidden layers, representing the current robot position and orientation, were used to select suitable navigation attractors in a sequence that leads to a goal location. All of this is achieved purely on an imitation learning basis, by generating examples of routes from a start location to a goal location.

RC networks can also be used in a generative way, by training the network to predict the sensory input nodes. These **Generative RC (GRC) networks**, when used in free-run mode, i.e., by shutting off the feedback from the environment and letting the network freely simulate its internal dynamics, it is possible to generate future scenarios, in terms of environmental perceptions and robot position, based on the behavior recorded in the training data. It was shown that a GRC network can simulate the future behavior of a mobile robot, which can potentially be used for path planning in a network which models multiple behaviors.

Learning procedures other than supervised learning have been shown to work well with RC networks. An unsupervised learning method which extracts slowly-varying features from an input signal, **Slow Feature Analysis** (SFA), when applied to the dynamic non-linear space of the reservoir, was shown to model **spatial encoding cells** which activate for multiple specific locations of the robot's environment. Another layer on top of the SFA layer im-

plementing sparse coding (with Independent Component Analysis) generates cells with very high place selectivity (in a way similar to hippocampal CA1 cells of rats).

As conclusion, RC proved to be a quite general method capable of modeling a diversity of navigation tasks under several types of learning paradigms (e.g, supervised and reinforcement learning), where hierarchical networks are able to model more complex implicit environment representations and navigation problems than single networks. *With the RC framework in this thesis, more abstract intelligent capabilities for generating efficient purposeful trajectories are made possible for mobile robots with restricted power and low computational demands, such as home cleaning robots.*

8.2 Discussion

An important concept for most experiments is **navigation attractor embedding**. These attractors are formed by reactive behaviors in the environment space which exhibit a pattern of sensory-motor sequences throughout the environment. Once projected into the dynamical system space of the reservoir by stimulating it with sensory signals, the navigation attractors become *embedded into the network* through the coupling of RC network (controller) and environment. Moreover, by using additional binary inputs representing multiple behaviors, these navigation attractors can be turned into sub-attractors in the dynamical system space, which ultimately boost the memory capacity of a single RC network, and can even be used by a generative network to autonomously simulate future paths taken by different behaviors for path planning.

The very general purpose RC method employed in this work has limitations with respect to generalization of trained networks to **behaviors not seen during training**. If an RC-based location detector is trained using a specific robot controller, it will only work during testing if a similar robot controller is used. This is because the dynamics which were induced in the reservoir by the sequence of sensory inputs is specific to the behavior of the robot. Moreover, the trained network will not recognize locations not seen in the training dataset, that is, it will **not be able to extrapolate to new loca-**

tions, unless an online learning method for new locations is devised (see next section).

Another point is that as the **randomness of the robot behavior** increases, it gets more difficult to learn implicit spatial representations of the environment and probably model the behavior itself as well. More random behaviors also require bigger reservoirs and more training data to make learning and generalization possible.

A related subject is how to **reduce the size of training datasets** in this thesis, composed of hundred thousands of samples. A detailed study is necessary, in this case, to point out the extent of dependence of the RC-based approach on the size of training data. With smaller datasets, regularization methods (Section 2.6) play a major role to guarantee generalization (i.e., good performance on unseen data). By reducing the number of training samples, the current approach will get concretely closer to the goal of *energy-efficiency* in navigation systems for small robots.

The performance of localization and navigation experiments can be very much improved by using **multiple different leak rates** in a single reservoir or in separate reservoirs (for achieving uncorrelated frequencies) mainly when the robot exhibits different speeds during navigation. This is an essential parameter to tune, which is correlated to the timescale of the input signal but also to the task at hand. For instance, a *localization reservoir* must operate at a slower timescale than a *navigation reservoir*, even though both reservoirs are stimulated with the same input. Leaky integrator units function as low-pass filters of the reservoir state and a similar effect can be achieved by **resampling the dataset** (with the advantage of reducing the size of the dataset, and the drawback of loosing details due to the averaging effect of downsampling).

8.3 Future work

Self-organized Learning of New Locations

Considering the architecture from Chapter 4, new locations not seen during training can not be detected by the RC network, as already mentioned. However, a method for solving this issue can easily be devised, at the expense of re-training the network. As soon as the

robot enters an unknown area or location, it is expected that all units in the readout output layer will show very low activity. This scenario of low activity may be used to label all samples generated during this period as a new location. These additional samples, concatenated to the original training dataset, can be used to re-train the RC network, characterizing a scheme of **autonomous learning of new locations**.

This autonomous process could even, in principle, be applied to training RC networks from scratch as the robot starts navigating, without the manual a priori labeling of data as was done in Chapter 4. This *autonomous self-organized mechanism* would take care of segmenting the real-time data into chunks to be classified as a single location, by discovering events such as, for instance, passing through the boundaries of rooms (which can be easily implemented by an extra RC network, as shown previously in this chapter) which share similar properties from the robot's perspective.

SFA for Goal-directed Navigation

In order to increase the autonomy of a navigation system, it is necessary to design **self-organized learning principles** so that effort on labeling training data is kept to a minimum. The idea is to move parts of or the whole architecture from a supervised learning mode to an autonomous learning mode. For instance, the hierarchical architecture of Chapter 5, composed of a localization module and a navigation module, could be modified so that at least the localization module is trained by an unsupervised learning method such as SFA, shown to construct implicit environment representations in a self-organized way (Chapter 7). As it was shown that trained SFA units exhibit activation patterns which are dependent on the robot path or behavior, in principle **SFA is suitable for goal-directed navigation** as the orientation of the robot matters significantly for learning the task (as predictions on both the previously visited and the current room were necessary to steer the robot to the destination room in Chapter 5).

SFA for Reinforcement Learning Navigation Tasks

It remains to be investigated how **SFA could be used in reinforcement learning (RL) navigation tasks** where behaviors are constantly evolving over learning iterations, as shown in Appendix A. At first, one would create an SFA layer between the reservoir layer and the output node representing the Q-value of the state-action pair, in the expectation that the implicit SFA representation would help to scale to bigger environments or speed up convergence of the learning process. However, as the SFA layer only performs a linear projection of the reservoir state space, it is more likely that this intermediate SFA representation does not help to improve the RL navigation task. An alternate and more interesting way of using SFA is by placing the SFA layer in between two reservoir layers. The second reservoir layer, receiving input from distance sensors as well as from the activations of SFA units, would be able, in principle, to learn much more complex relationships between the current context of the robot in its environment and the desired action. This new architecture is analogous to the hierarchical architecture of Chapter 5, in which a navigation reservoir is driven by the activities generated by the localization module. In the same way, it is expected that SFA units could drive the operating point of the reservoir state space to different sub-attractors, and in this way increase the memory capacity and the possibility to use more complex environments.

Planning with Hierarchical Networks

A quite interesting system can be designed by combining the generative RC network of Chapter 6 and the hierarchical network used for goal-directed navigation in Chapter 5. In this way, a **generative hierarchical network**, which predicts also sensory nodes, can be used in free-run mode to simulate paths throughout complex environments towards a given destination room. The autonomous simulation which runs disconnected from the environment would create predictive sequences of perceptions, robot positions, and actuators according to an internal dynamics induced by the imitation learning process. Not only one, but multiple trajectories to destination rooms could be simulated without actually driving the robot, cre-

ating possibilities for deliberative path planning processes based on a very low-dimensional raw sensory space given by the few distance sensors of the robot.

Generative RC networks for Path Planning based on Rewards

Generative RC networks could also be used in deliberative navigation systems which choose a **rewarded sequence of navigation attractors** to perform. This architecture, similar to the one in Chapter 6, would have additional inputs representing behaviors and an output node modeling the immediate reward. The training process would collect data by randomly sequencing navigation attractors (corresponding to these extra inputs) throughout the environment and only rewarding the desired sequence through space, by teacher forcing the output node modeling the reward. This generative architecture, trained in a supervised way, can be used in the testing phase to discover what sequence of behaviors should be chosen to reach the rooms where the robot previously received rewards, by performing a search over all behaviors using the generative network in free-run mode. This ultimately leads to the formation of planning-like capabilities in a reinforcement learning task which is modeled by a supervised learning algorithm (which in principle also requires the labeling of the rooms of an environment).

Explicit Map Generation with Extended RC-SFA architecture

Considering the RC-SFA architecture from Chapter 7, future research could be done on **generative models** which predict the robot perceptual input given the activation of the ICA layer. This can be implemented in a supervised fashion using a reservoir with the location as inputs (ICA units) and the sensory input as desired output, as described in Chapter 6. Although the training of this generative model is supervised, the whole learning process is still unsupervised, because no labels for the locations are needed since they emerge in a self-organized way with SFA and ICA. By using this generative architecture, it is possible not only to *generate explicit maps of the environment* given the ICA activation but also to create **fault-tolerant location detection** in the ICA layer in situations where sensors are broken or faulty by predicting this missing

information (see Section 6.5).

Towards biologically-inspired SLAM

In the context of robot navigation and localization, future work include the integration of the allothetic representation of the RC-SFA architecture from Chapter 7 with idiothetic signals such as the robot position estimated by a path integration module (odometry). Using this setup, the new architecture could be compared to a basic form of **biologically-inspired SLAM** (as ratSLAM in Milford, 2008).

Deep Hierarchies with RC-SFA

In principle, the proposed RC-SFA architecture from Chapter 7 would **scale to larger environments**, but likely not to very random robot behaviors due to the limited memory capacity of reservoirs (unless sub-attractor switching mechanisms are used to increase total memory as seen in Chapter 5). Alternate or additional attempts to scale to more complex environments and richer sensory data (e.g., a camera) are using *deeper hierarchies* with receptive fields or even *multiple reservoirs* situated at different layers in the hierarchy.

A

Reinforcement Learning in non-Markovian Navigation Tasks

Autonomous robot navigation in partially observable environments is a complex task because the state of the environment can not be completely determined only by the current sensory readings of a robot. For instance, the distance sensors provide only a local view of the environment which can not always be used to infer the absolute position of the robot in it. Temporary signals or commands given to the robot which are correlated with future motor responses (as in delayed response tasks) are another cause of partial observability. These problems are typically modeled as Partially Observable Markov Decision Process (POMDP), where the underlying state can not be directly observed by an agent, and are usually intractable to solve exactly. In this chapter, RC is used to approximate the state-action value function in non-Markovian navigation tasks. By using a policy iteration framework, where an alternating sequence of policy improvement (samples generation from environment interaction) and policy evaluation (network training) steps are performed, the system is able to shape navigation attractors so that, after convergence, the robot follows the correct trajectory towards the goal. The experiments are accomplished using an e-puck robot extended with 8 distance sensors in a rectangular environment with an obstacle between the robot and the target region. The task is to reach the goal through the correct side of the environment, which is indicated by a temporary stimulus previously observed at the beginning of the episode. We show that the reservoir-based system (with short-term memory) can model these navigation attractors, whereas a feedforward network without memory fails to do so.

A.1 Introduction

Autonomous robot navigation tasks frequently need to be solved under the assumption that the state of the environment is partially observable, where the robot does not know the complete state of the environment just by reading its current sensors, such as its position in the environment. This is usually referred in the literature as a partially observable Markov decision process (POMDPs). Thus, the navigation system requires additional sources of information for building the complete state of the environment, such as using wheel encoders to estimate the traveled distance and the current robot position. Instead of explicitly modeling odometry, Chapters 4 and 7 show that RC networks can detect the location of the robot in a supervised or unsupervised way.

Reinforcement Learning (RL) rules are based on a reward signal $r[t]$, which represents the outcome (i.e., success or failure) of a learning trial. In immediate reinforcement learning, the reward signal is given at every timestep and is usually computed as the *distance to the goal*. On the other hand, for RL methods with delayed reward signals the outcome is given only at the end of a trial. Whereas in the former case hints are given in every moment, the latter case defines a very less informative reward function which can delay the learning process, although it is more biologically plausible. RL algorithms such as *Q-learning* (Sutton and Barto, 2000) learns an action value function $Q(s, a)$ which indicates the utility of an action a in a state s. It tries to maximize the mean expected future reward by using a *value iteration update* rule. Whereas these rules are applied in an online fashion, in fitted Q iteration (Riedmiller, 2005) the $Q(s, a)$ is learned in batch mode with supervised learning techniques such as linear regression and artificial neural networks.

RC-based systems are usually trained in a supervised way. Some works in the literature use RC in a Reinforcement Learning framework to model partially observable environments (Bush, 2008; Szita et al., 2006). In Szita et al. (2006), an RC network with multiple outputs in the readout layer, each one corresponding to the value of a discrete action, are trained online via a SARSA update (Sutton and Barto, 1998) rule, whereas in Bush (2008), an RC network with one output representing the value function for a state-action input

pair is trained in an iterative batch-mode way. Both works consider partially observable environments, but while Szita et al. (2006) applies to discrete-world tasks, Bush (2008) consider more complex control tasks such as the acrobot swing-up task (Sutton and Barto, 1998).

Function approximators are typically used to model the state-action value function in complex problems where the state space is continuous, and in most cases a ϵ-greedy policy on the value function is used for selecting the optimal action, where ϵ defines the probability of selecting random actions for exploration of the state space. In particular, sample-based methods like fitted Q iteration (Riedmiller, 2005) and least-squares policy iteration (Lagoudakis and Parr, 2003) are efficient batch-mode training methods for modeling the state-action value function using function approximators. They collect samples as tuples

$$(s_t, a_t, r_t, s_{t+1}),$$

where s_t is the state at time t; and s_{t+1} is the next state after executing action a_t on state s_t and receiving the reward r_t, by interacting directly to the environment using either totally random actions or a ϵ-greedy policy. The dataset composed of samples are iteratively used for batch training.

This chapter uses a similar approach as Bush (2008). In that work, an RC network is used to model non-Markovian environments in control tasks such as the mountain car problem and the acrobot swing-up task. It uses a reservoir to convert a non-Markovian state to an approximate Markovian state representation embedded in the high-dimensional state-space of the reservoir.

Specifically, the current chapter uses an RC network to model partially observable environments in robot navigation problems. In Chapter 3, navigation attractors were learned via imitation learning. Here, multiple navigation attractors are iteratively shaped throughout the evolutionary process of trial and error, until a suitable set of task-achieving attractors or behaviors is found. The experiments are performed using an e-puck robot with 8 distance sensors in a rectangular environment with an obstacle between the robot and the target region. The task of the robot is to reach the goal through

the correct side of the environment. A temporary stimulus at the beginning of the episode indicates which side of the environment the robot should use. So, to successfully perform the task, the navigation system must have some sort of short-term memory.

In the proposed setup, the RC network is iteratively trained in batch mode to approximate the state-action value function given as input 8 distance sensors, a discrete action and an additional input which simulates the temporary stimulus. The discrete action is modeled by three *motor primitives* (left, right and forward) for moving the robot in the environment. Experimental results show that after an initial exploration period and a sequence of policy improvements steps, the robot is able to navigate to the goal circumventing the obstacle through the correct side (determined by the initial stimulus) in a quasi-optimal trajectory. It is also shown that the fading memory of the reservoir is essential to model these context-dependent navigation attractors.

A.2 Methods

A.2.1 Reservoir Computing for Q-value Approximation

In fitted Q iteration (Ernst et al., 2005), samples in form of tuples

$$(s_t, a_t, r_t, s_{t+1}), \quad t = 1, \cdots, I,$$

are generated from interaction with the environment and collected in a training dataset. Training the system is done offline using the collected samples under a supervised learning framework: usually, a regression algorithm is used to learn the state-action value function, by defining the input and the desired output as follows:

$$\mathbf{u}[t] = (s_t, a_t), \tag{A.1}$$

$$\hat{y}[t] = r_t + \gamma \max_a \hat{Q}_{N-1}(s_{t+1}, a), \tag{A.2}$$

where: s_t, a_t and r_t are the state, action and reward at time t, respectively; N is the iteration of the training process; and γ is the discount factor. Using the dataset of input-output pairs $(\mathbf{u}[t], \hat{y}[t])$, the function $\hat{Q}_N(s, a)$ is induced with a regression algorithm.

In this chapter, an analog sigmoidal RC network or Echo State Network (ESN) is used to model the critic, that is, the Q-value (Sutton and Barto, 1998) function, in non-Markovian environments. Given a partially observable state vector $\tilde{\mathbf{s}}$ and an action a as input, the goal is to approximate the expected future sum of rewards, the Q-value for the pair $(\tilde{\mathbf{s}}, a)$, using an RC network as approximation method. The randomly generated reservoir can convert non-Markovian state-spaces into Markovian state-spaces due to its characteristic fading memory of previous inputs. This method is similar to fitted Q iteration (Ernst et al., 2005; Riedmiller, 2005) and least squares policy iteration (Lagoudakis and Parr, 2003) in that it is based on batch offline training and approximates the value function in an iterative way.

In Bush (2008), the RC network is used in reinforcement learning control tasks such as the mountain car problem and the more complex acrobot swing-up task. The input to the reservoir is a vector $\mathbf{u}[t]$ composed of a partially observable state $\tilde{\mathbf{s}}$, such as the position of the car or the joint angles of the acrobot (so, excluding the velocity component), and an action a, and the only output is trained to approximate the state-action value function.

As $\hat{Q}(\tilde{s}, a)$, the desired output $\hat{y}$, can be approximated by a sum of future rewards over a finite time horizon h (Bush, 2008), equations (A.1) and (A.2) can be rewritten, in the case of a non-Markovian environment:

$$\mathbf{u}[t] = (\tilde{s}_t, a_t), \tag{A.3}$$

$$\hat{y}[t] \approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^h r_{t+h} \tag{A.4}$$

The training is accomplished in an iterative way and consists of a sequence of policy improvement and policy evaluation steps (see Fig. A.1). During policy improvement, new samples $(s_t, a_t, r_t), t = 1, ..., I$ are generated using a ϵ-greedy policy and the trained architecture. I is the number of samples generated during one iteration of the policy improvement stage, which is set to $I = 1000$. During policy evaluation, the training input-output pairs $(\mathbf{u}[t], \hat{y}[t]), t = 1, ..., E$ are generated using (A.3) and (A.4), respectively, and the RC network is trained on a subset of the dataset generated through interaction with the environment. This subset corresponds to a slid-

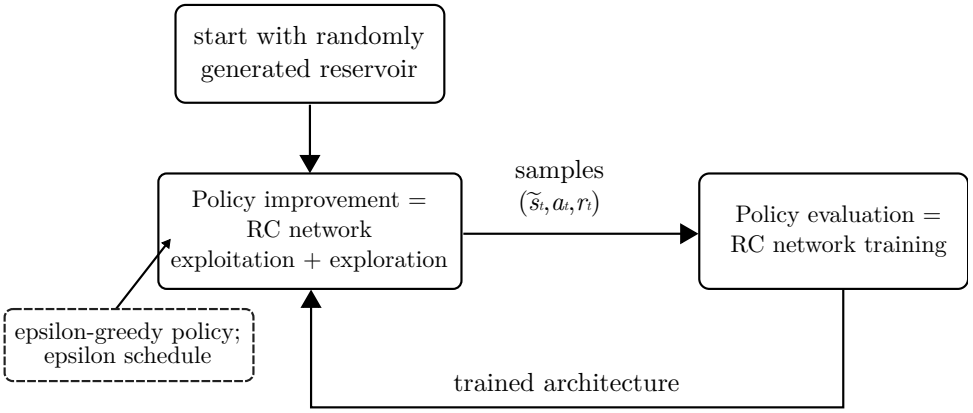


Figure A.1: Approximate Policy Iteration: Policy improvement + Policy evaluation. The iterative policy learning consists of: generation of samples by interacting with the environment using a ϵ-greedy policy and the trained architecture (policy improvement); and of training the architecture (in this case, the RC network) to approximate the state-action value function with a regression algorithm using the dataset generated during policy improvement. $\tilde{s}$ is a partially observable state, characterizing a non-Markovian task which should be handled by the RC network.

ing window of samples of size E, such that only the most recent $E = 40.000$ samples are used for training. During the iterative policy learning process, the ϵ-greedy policy follows a learning schedule where the exploration is intense at the beginning of the process and monotonically decreases towards the end of the experiment. This is accomplished by varying ϵ according to a predefined schedule (Bush, 2008) (given in Section A.3.1).

The RC network used in this chapter is shown in Fig. A.2. The equations of the model and its training method, linear regression, are described in Chapter 2. Specifically, it is considered in this chapter the reservoir state update equation with leaky-integrator neurons given by (2.9), while the equation for the readout output $y[t]$, which models the state-action value function in this chapter, is given by (2.6).

The **exploitation of the RC network** in Fig. A.2 for the

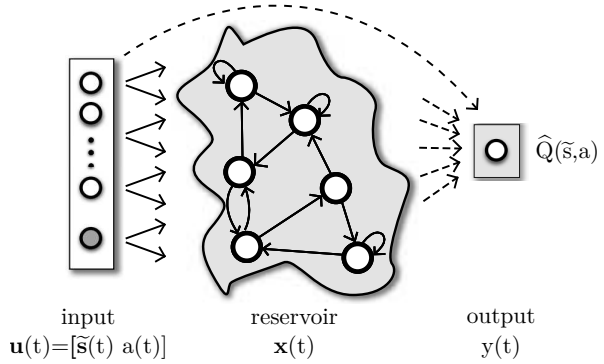


Figure A.2: Reservoir Computing network as a function approximator for modeling the *critic* (i.e., state-action value function) in reinforcement learning tasks with partially observable environments. Solid lines represent connections which are fixed. Dashed lines are the connections to be trained.

control task is based on the following equations:

$$a_{opt}[t+1] = \arg\max(y[t+1]) \tag{A.5}$$

$$a_{opt}[t+1] = \arg\max\left(\begin{bmatrix}\mathbf{W}_{\mathrm{r}}^{\mathrm{o}} & \mathbf{W}_{\mathrm{i}}^{\mathrm{o}}\end{bmatrix}\begin{bmatrix}\mathbf{x}_a[t+1]\\ \tilde{\mathbf{s}}[t]\\ a\end{bmatrix}\right), \tag{A.6}$$

where $\mathbf{x}_a[t+1]$ is an internal reservoir state which is *dependent on the action a tested during the application of* $\arg\max$:

$$\mathbf{x}_a[t+1] = f\left(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}\mathbf{x}[t] + \mathbf{W}_{\mathrm{i}}^{\mathrm{r}}\begin{bmatrix}\tilde{\mathbf{s}}[t]\\ a\end{bmatrix}\right).$$

This means that the reservoir state is *freezed* at timestep t, and to choose the optimal action, the arg max function runs the reservoir for each value of action a always starting at the same reservoir state $\mathbf{x}[t]$ from timestep t. For instance, Fig. A.3 shows how the reservoir state evolves over time by using the argmax function on three possible values for action a $(-1, 0, 1)$.

A.2.2 Robot Model and Motor Primitives

The Webots extended e-puck robot model shown in Fig. 5.1 and described in Section 5.3.1 or Appendix B.2 is used in this chapter.

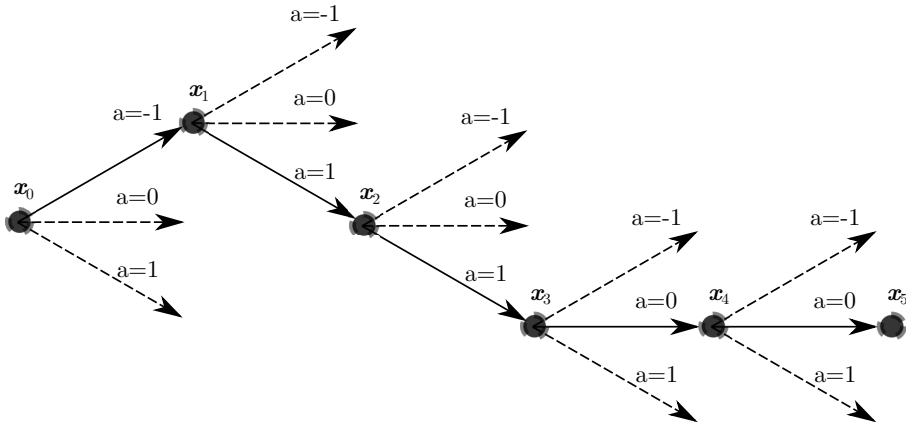


Figure A.3: Evolution of the reservoir state $\mathbf{x}[t]$ over time as the operator arg max is applied to the RC network from Fig. A.2. Dashed lines represent reservoir states which generated suboptimal paths during the application of arg max operator. The real path followed by the reservoir is given by solid lines.

Next, the changes of the e-puck model valid for this chapter are presented. The extension turret has distance sensors capable of measuring distances in the interval (0, 30] cm. The noise on sensors is drawn from a normal distribution $N(0, 3)$ cm. The robot has 2 stepper motors with maximum speed of 1000 steps per second, which are steered by the following **3 motor primitives** or basic behaviors in the low-level control module: forward (left wheel: 500 steps/s; right wheel: 500 steps/s), left (left wheel: 250 steps/s; right wheel: 500 steps/s), and right (left wheel: 500 steps/s; right wheel: 250 steps/s). These motor primitives are executed for a period of 11 timesteps in the simulator (704 ms). See Fig. A.4 for a graphical representation of the trajectories given by each of the motor primitives. It is relevant to observe that each primitive is inherently stochastic once the robot wheels can not reproduce the same trajectory due to non-systematic noise originated from wheel-slippage or irregularities of the floor.

The motor primitives are designed to simplify the control task, by reducing the action space to 3 discrete actions.

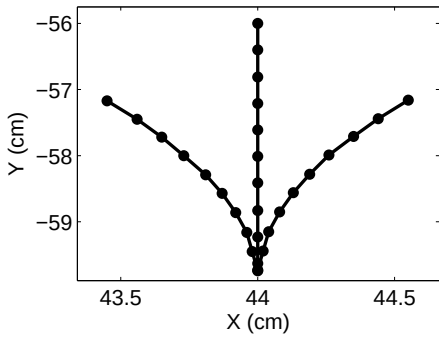


Figure A.4: Motor primitives or basic behaviors: left, forward and right.

A.3 Experiments

The robot task is to learn context-dependent navigation attractors in a partially observable environment. The environment is a rectangular arena with an obstacle between the robot and the goal location, as it can be seen in Fig. A.5(a). During a simulation experiment, each episode starts with the robot located in the upper part of the room with position randomly chosen from a small interval defined by the solid rectangle in Fig. A.5(b); the initial orientation of the robot is South, with small uniform noise added in the range $[0, 1.2]$ degrees. The robot is controlled according to a ϵ-greedy policy. The architecture is trained using the scheme depicted in Fig. A.1 and explained in Sections A.2.1.

The task of the robot in this environment consists of navigating to the goal location, given by the light blue dashed box in Fig. A.5(b), through the left or right part of the environment, shown by black and gray dashed rectangles in the same figure, depending on a previously received stimulus from the environment. This temporary stimulus can be implemented through the presence/absence of an object in the environment, the on/off of a light source, or the existence/absence of a sound. In the current experiments, this is simply implemented as an additional input signal to the reservoir which is 1.5 whenever the trajectory towards the goal should be done via the left side and -1.5 when the this trajectory should be performed via the right side. This extra signal is present for 2.1 s in the beginning of each episode, during which the robot is not able to

Table A.1: Parameter configuration for RC network

Number of input channels	$n_i = 10$
Input connection fraction	$c_i^r = 0.5$
Input scaling	$v_i^r = 0.14$
Input downsampling	$d_t = 1$
Input to output connections	yes
Bias connection fraction	$c_b^r = 1$
Bias scaling	$v_b^r = 0.2$
Reservoir size	$n_r = 400$
Reservoir connection fraction	$c_r^r = 0.1$
Spectral radius	$\rho(\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}) = 0.9$
Leak rate	$\alpha = 0.1$
Number of output channels	$n_o = 1$
Output feedback to reservoir	no

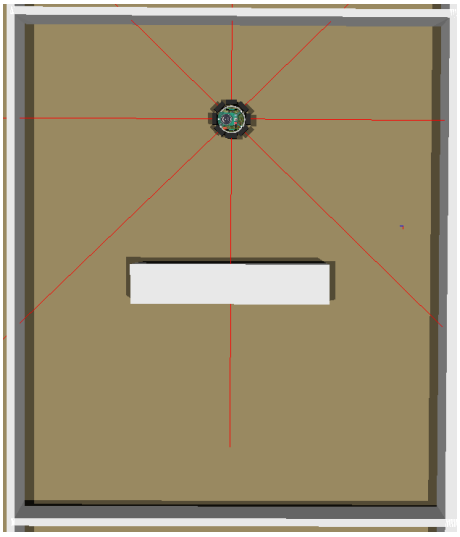
go left or right but only slowly forward (meant not to bias learning). After the initial period of 2.1 s, this extra input becomes zero.

One episode is finished whenever the robot reaches the goal performing the correct trajectory, hits against an obstacle, or when the the length of the episode is greater than 60 timesteps. The reward r_t is always -1, unless the robot is at the goal location, when $r_t = 0$. When an episode ends, the input and desired output can be computed according to equations (A.3) and (A.4).

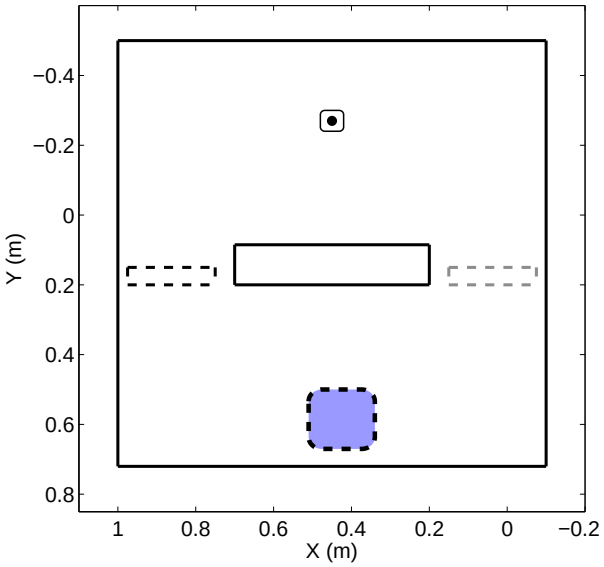
A.3.1 Settings

Table A.1 shows the parameter configuration for the RC network. The inputs $\mathbf{u}$ to the network are 8 frontal distance sensors, scaled to the interval [0,1], an action $a \in \{-1, 0, 1\}$ and an additional input for the temporary stimulus.

The ϵ parameter for the policy, which corresponds to the probability of selecting random actions at each time step, is selected from an arbitrarily chosen vector $[0.9, 0.8, 0.6, 0.5, 0.4, 0.3, 0.1, 0.01]$, similarly to Bush (2008). The particular timesteps in which ϵ changes follows a learning schedule chosen as [40, 140, 190, 220, 240, 260, 310, 330]$*10^3$ timesteps. This means, for instance, that during the first 40.000 timesteps, $\epsilon = 0.9$. The finite time horizon in (A.4) is $h = 40$. The discount factor is $\gamma = 1$, which defines a shortest-path



(a) 3D environment and e-puck robot



(b) Corresponding Map in 2 dimensions

Figure A.5: Rectangular environment with an obstacle between the robot and the goal location. (a) 3D environment in Webots, with the e-puck robot in the upper part. (b) Representative map of the environment in two dimensions. The box with a point inside represents the possible starting positions for the robot (randomly chosen), while the black and gray dashed rectangles represent the possible circumvention areas (dependent on the initial transient stimulus) which the robot has to use to reach the goal, represented by dashed box in light blue color.

problem.

The regression learning procedure for the reservoir architecture is executed every 1.500 timesteps considering the last $E = 50.000$ generated samples as learning window. These samples used for learning are generated from the interaction of the reservoir with the environment, while samples resulting from random actions are not taken into account during learning.

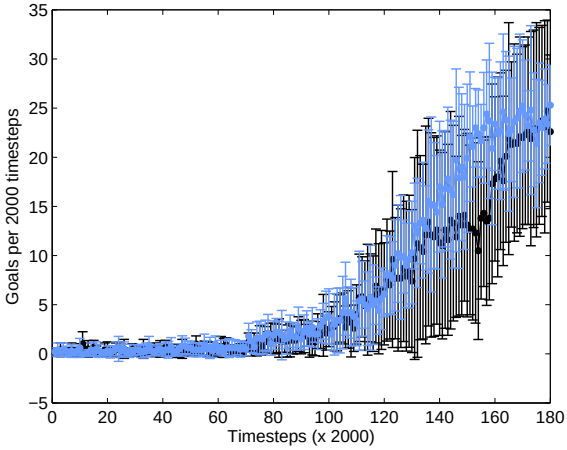
A.3.2 Results

In order to evaluate the proposed robot navigation task using the RC network, the mean number of goals achieved per 2×10^3 timesteps considering *left and right trajectories* separately is shown in Fig. A.6(a). As time evolves, exploration decreases and the number of goals achieved via left and right trajectories (represented by black and blue lines, respectively) increases, which shows the capability of the architecture to learn short-term temporal dependencies in robot navigation tasks.

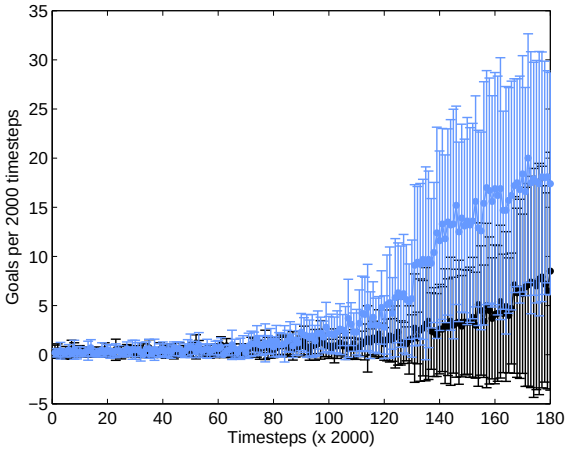
In Fig. A.6(b), the mean number of achieved goals is computed using a memoryless architecture, implemented by simply setting the reservoir weights $\mathbf{W}_{\mathrm{r}}^{\mathrm{r}}$ to zero. It is possible to observe that the system does not learn the task correctly, preferring the *right trajectory* over the *left trajectory* in most of the experiments because the number of goals increases for the right navigation attractor (in blue) and decreases for the left attractor.

A single RC network can model multiple navigation attractors in a reinforcement learning task. These attractors, in the context of reinforcement learning, are dynamic, because the agent-environment interaction changes over time. Fig. A.7 and A.10 show how these dynamic attractors evolves during the learning process. In the beginning, the two navigation attractors are not well formed, also because exploration is very high. In that stage, the system performs several possible trajectories due to random actions. As the simulation advances, the dynamic attractors are shaped so that the robot reaches the goal location performing a trajectory which is dependent on the initial temporary stimulus given at the beginning of the run.

Fig. A.8 shows the principal components resulting from applying PCA on the reservoir states for the last episodes of simulation of Fig. A.7. The principal component 3 encodes information used to



(a) RC network



(b) The same network but without recurrent connections

Figure A.6: Average number of goals achieved per two thousand timesteps for 10 simulation experiments. The black lines represent the goals achieved via the *left trajectory*, while the blue lines represent the goals achieved via *right trajectory*. Error bars represent the standard deviation between runs. (a) Using the reservoir architecture presented in this chapter. (b) Using the same architecture, but without internal memory by setting $\mathbf{W}_{\text{res}}^{\text{res}} = 0$.

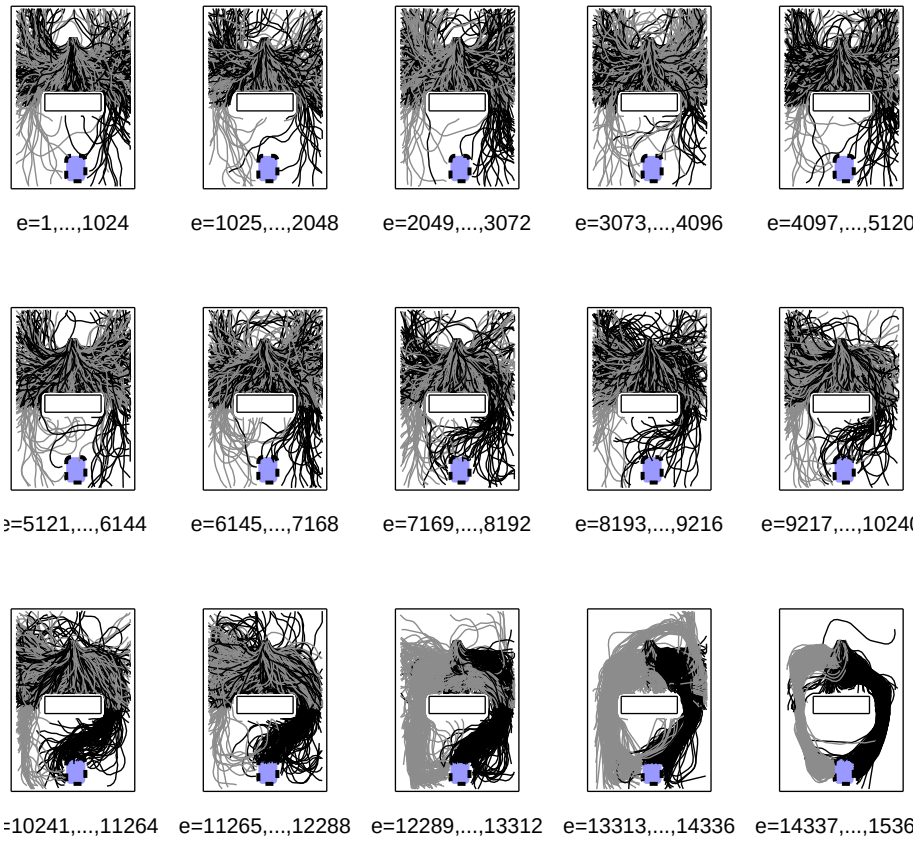


Figure A.7: A sequence of robot trajectories as learning evolves, using the RC network. Each plot shows robot trajectories in the environment for several episodes during the learning process. In the beginning, exploration is high and several locations are visited by the robot. As the simulation develops, two navigation attractors are formed to the left and to the right so that the agent receives maximal reward.

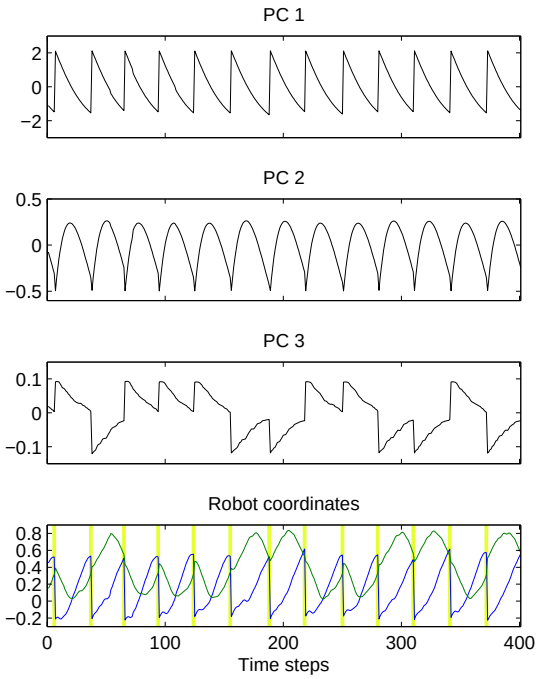


Figure A.8: Three principal components (PC) over time after applying PCA on the reservoir states, at the end of the simulation corresponding to last episodes in Fig. A.7. The bottom plot shows the robot coordinates x, y over time in the environment. The yellow vertical lines delimit different episodes. These plots were made disregarding the initial timesteps where the temporary stimulus is given, i.e., those initial timesteps were removed. The PC 3 encodes information used to follow the correct trajectory (left or right), thus forming a short-term memory responsible for holding the initial stimulus.

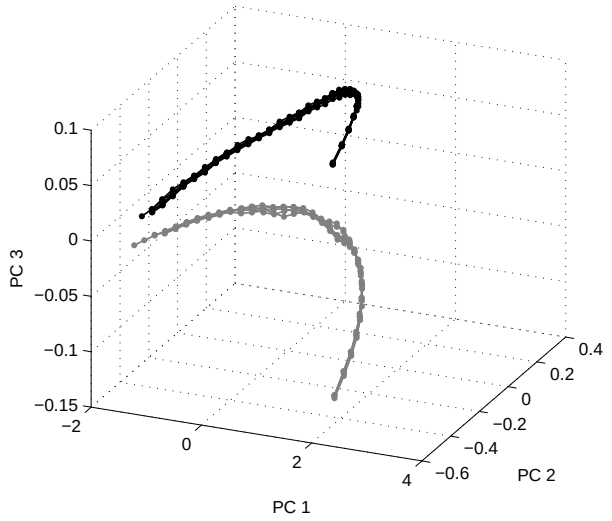


Figure A.9: *Sub-space attractors* in the reduced dynamical system space for *left and right* navigation trajectories. The plot shows a 3D state space of the principal components of Fig. A.8, where gray and black lines represent different (left and right) trajectories in the environment, which are dependent on the previously received transient external stimulus.

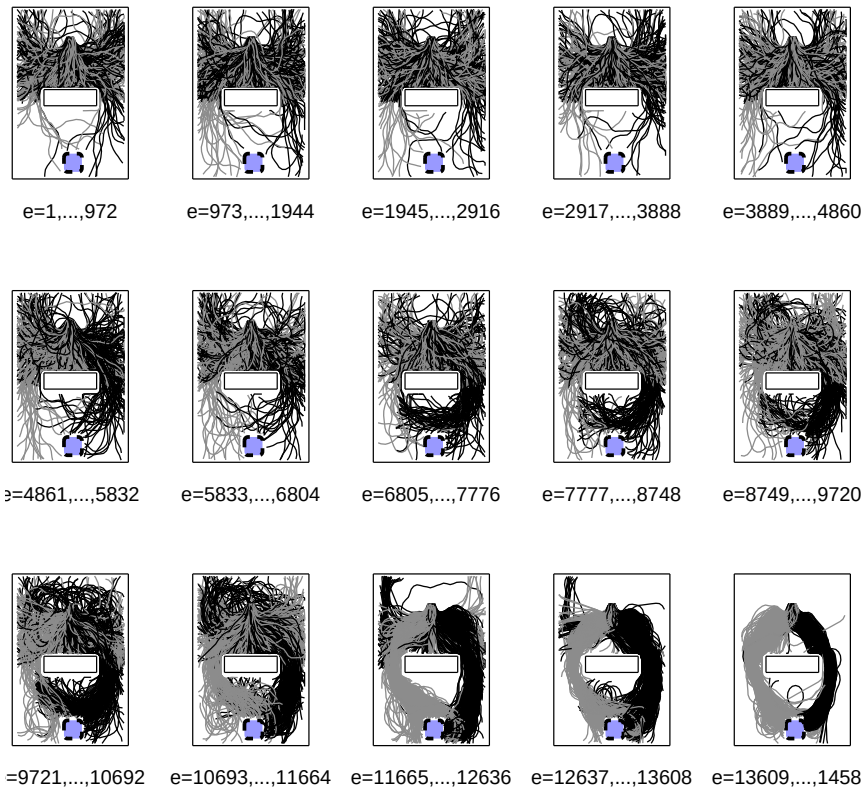


Figure A.10: Another sequence of robot trajectories as learning evolves, using the RC network. Explanation as in Fig. A.7.

follow the correct trajectory at the left or right side, thus forming a short-term memory responsible for holding the initial temporary stimulus. Fig. A.9 shows that, after convergence of the learning process, the principal components form different trajectories (or subspace attractors) in the state space according to the past stimulus given at the beginning of the episode.

Without the fading memory of the reservoir, it is not possible to learn these navigation attractors correctly, because a memoryless architecture does not hold the temporary stimulus for future moments.

A.4 Conclusion

This chapter has shown that an RC network can be used to model the state-action value function in non-Markovian navigation tasks. The training procedure is similar to Bush (2008). In this chapter, I relate that method to a policy iteration framework (policy improvement + policy evaluation) (Lagoudakis and Parr, 2003), consisting of an alternating sequence of simulation experiments for samples generation and regression learning events.

The reservoir projects a non-Markovian input into a high-dimensional non-linear state-space with temporal dynamics. This dynamic state-space converts the non-Markovian environment into a Markovian problem, as it automatically takes the history of the input stream into account.

The non-Markovian task of the robot in this chapter is similar to the T-maze task (Chapter 3) (Ulbricht, 1996) which requires a temporal association of a past stimulus and a future delayed reponse. However, the work in this chapter is more general since it requires to learn the shortest path to a goal via a complex sensory-motor coupling which depends on the initial temporary stimulus. In the T-maze task the robot is confined to a very tight environment in most scenarios and robot controllers for this task could exploit this characteristic to facilitate the navigation task. On the other hand, in this chapter, navigation attractors are formed by a non-trivial sequence of basic behaviors (or motor primitives) which have to control the trajectory of the robot towards the goal. It is a control

task where wheel slippage may interfere negatively the trajectory. Thus, the navigation system (RC network) must learn to perform robust behaviors for task achievement.

There are several possibilities for continuation of this work. First, experiments can be done with extended navigation tasks where the robot navigates between multiple rooms of an environment, which would allow to evaluate how the architecture scales to more complex tasks. It would also be interesting to investigate how many navigation attractors a single reservoir network can learn and how this is related to the size of the reservoir.

In the current chapter, the RC-based system learns to control by generating a sequence of motor primitives which can lead to collisions. An alternate method is to use as the motor primitives low-level controllers such as Braitenberg vehicles (as in Chapter 5) which automatically avoids obstacles. In this case, the search space for navigation attractors would be hugely reduced, but not necessarily to the desired sub-space of attractors. For instance, this alternative would be useful for confined environments, such as the T-maze, but not necessarily for environments with open space which require shortest-path trajectories to the goal. Experiments using Braitenberg vehicles as motor primitives in the navigation task from this chapter indicate that it is not possible to converge to shortest path trajectories, once the controllers tend to go torwards a wall and follow the wall until it reaches the goal.

Finally, other techniques such as Slow Feature Analysis (Wiskott and Sejnowski, 2002) (tackled in Chapter 7) could be investigated in order to evaluate whether it can be used to autonomously form spatial environment representations during the reinforcement learning process. In this case, the idea is the possibility of navigation in large environments due to the feature extraction and localization properties given by SFA.

B
Robot Models

Throughout this thesis, two robot simulators are used for accomplishing experiments: SINAR and Webots. The real e-puck robot as well as its simulated version in Webots are used in experiments.

The data originated from the robot such as distance sensors and actuators as well as from the environment (e.g., robot position) are collected from the robot simulators or from the real e-puck robot. These data are used to train and test RC networks in a Matlab environment using the RCT Toolbox [1] (Verstraeten et al., 2007).

SINAR is described in Section B.1, whereas the (simulated and real) e-puck robot model is presented in Section 4.2.1.

B.1 SINAR

B.1.1 Description

SINAR is a 2D autonomous robot simulator introduced in Antonelo et al. (2006), where the mobile robot (Fig. B.1(a)) interacts with the environment by distance and color sensors; and by one actuator which controls the movement direction (turning). Most of the experiments in this thesis consider models of environment and robots in this simulator as follows.

The environment of the robot is composed of several objects, each one of a particular color. Particularly, obstacles are represented

[1] This is an open-source Matlab toolbox for Reservoir Computing which is freely available at http://www.elis.ugent.be/rct

Table B.1: SINAR robot models

	SINAR 1	SINAR 2	SINAR 3
No Dist. Sensors	17	3	7
No Color Sensors	17	3	7
Range of Dist. Sens.	$300d.u.$	$300d.u.$	$300d.u.$
Noise on sensors	$N(0, 3d.u.)$	$N(0, 15d.u.)$	$N(0, 15d.u.)$
Speed	$0.28d.u.$	$[0, 17]d.u.$	$[0, 17]d.u.$

by blue objects whereas targets are given by yellow objects. The robot model has 17 sensor positions [2] distributed uniformly over the front of the robot, from -90° to +90°. Each position holds two virtual sensors for distance and color perception. The distance sensors are limited in range such that they saturate for distances greater than 300 distance units ($d.u.$), and are noisy - they exhibit Gaussian noise $N(0, 0.01)$ on their readings. A value of 0 means near some object and a value of 1 means far or nothing detected. At each iteration the robot is able to execute a direction adjustment to the left or to the right in the range [0, 15] degrees and the speed is equal to 0.28 distance units $(d.u.)/s$ [3].

The 3 different robot models used in the SINAR simulator are summarized in Table B.1.

B.1.2 Robot Controller

The SINAR controller, described in Antonelo et al. (2006), is a reactive intelligent navigation system made of hierarchical neural networks which learn by interaction with the environment. After learning, the robot is able to efficiently navigate and explore environments, during which the input signal $\mathbf{u}[n]$ for equation (2.1) is built by recording the 17 distance sensors of the robot.

[2] The number of sensors may be different depending on the experiment. In this case, the differences will be described with the experiment.

[3] The robot speed can also be modified depending on the experiment.

B.2 E-puck

B.2.1 Description

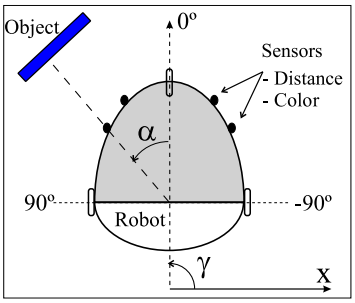
The e-puck (Mondada, 2007) is a small differential wheeled robot which was built primarily for education purposes, but has been largely adopted in research as well. The mobile robot has a diameter of 7 cm and is equipped with 8 infra-red sensors which measure ambient light and proximity of obstacles in a range of $4cm - 5cm$ originally, which effectively restricts the ability to read distances to obstacles. Because of this, an extension turret for the real e-puck robot has been built with 8 longer-range infra-red sensors capable of measuring distances in the interval [4-30] cm (see Fig. B.1(d)). The actuators of the robot are 2 stepper motors (steps/second), where the maximum speed is 1000 steps per second.

The experiments in this thesis are accomplished with a simulated version of the e-puck robot as well as with the real e-puck robot. For simulations, we use the Webots software (Michel, 2004) which provides a physics model of the e-puck robot (Fig. B.1(b)), i.e. the simulator detects collisions and simulates physical properties of objects, such as the mass, the velocity, the inertia, the friction, the spring and damping constants, etc.

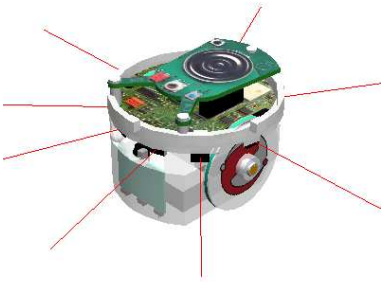
In the simulator, the original e-puck model (Fig. B.1(b)) as well as an extended e-puck model (Fig. B.1(c)) are used. These two robot models can be organized in 4 configurations according to the experiments in this thesis which consider different types of sensors:

- e-puck 1: original e-puck with a short-range sensor model which measures distances up to $5cm$;
- e-puck 2: original e-puck with a longer-range sensor model which reads distances up to $15cm$;
- e-puck 3: extended e-puck with a longer-range sensor model which reads distances in the interval $[5 - 80]cm$;
- e-puck 4: extended e-puck with a longer-range sensor model which reads distances up to $80cm$;

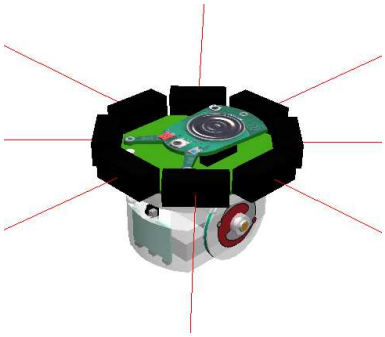
Table B.2 summarizes these 4 configurations or models. The velocity of the simulated robot is limited to [0.6, 3] cm/s.



(a) SINAR



(b) **Original** Webots e-puck



(c) **Extended** Webots e-puck

(d) **Extended Real** e-puck

Figure B.1: Robot models used throughout this work. (a) SINAR robot model with distance and color sensors (usually in number of 17 of each type) positioned in the frontal part of the robot (-90 to 90). (b) E-puck robot from Webots simulation environment. (c) E-puck robot from Webots simulation environment, extended with simulated longer-range infra-red sensors. (d) Real e-puck robot extended by an additional turret containing 8 infra-red sensors capable of reading distances from 4 cm to 30 cm.

Table B.2: Webots e-puck robot models

	e-puck 1	e-puck 2	e-puck 3	e-puck 4
Range of Dist. Sens.	$[0-5]cm$	$[0-15]cm$	$[5-80]cm$	$[0-80]cm$
Robot Model (Fig. B.1)	original	original	extended	extended
Chapter	4	4	5	A
Task	event detection	localization	localization & navigation	navigation

B.2.2 Robot Controller in Webots

The controller for the simulated e-puck robot is made of a simple algorithm which follows points from a predefined trajectory in the environment. Its speed can be 3 cm/s, 1.25 cm/s or 0.63 cm/s.

B.2.3 Robot Controller in Real Environments

For recording datasets containing the robot's sensor readings, a controller written in Matlab steers the e-puck robot through a Bluetooth connection. This controller performs basic wall following throughout the environment and it switches randomly to left or right wall following with a certain probability τ[4]. When the robot switches from right to left wall (or vice-versa), it may generate ellipsoid trajectories inside a room until it finds a wall to follow. Thus, the robot may stay navigating inside a particular room for a random time interval. The results shown in this chapter considers that $\tau = 0.03$, which practically means that there is a probability of circa 60% for inverting the direction of movement while the robot is navigating inside one of the rooms.

One iteration, for reading the distance sensors as well as for motor actuation, lasts 200 ms. The speed actuator is limited to the interval $\pm[15, 385]$ steps/s (or $\pm[0.198, 5.08]$ cm/s).

The eight distance sensors are sequentially read in groups of 2 while the robot is moving, that is, there are 4 cycles of sensors reading, where each cycle corresponds to 2 simultaneous readings. Considering an acquisition time of 25 ms on average for a cycle, the total time spent on sensor reading is between 100 and 120 ms. Any resulting inconsistencies from this sequential sensor reading during robot movement are not corrected, so that learning has to cope with this additional problem.

The input signal $\mathbf{u}(t)$ for the reservoir equation in (2.1) is built by recording the eight distance sensors during robot navigation and scaling them to the interval $[0, 1]$. For analysis purposes, the robot position and orientation are estimated using pictures taken from a fish-eye camera placed on a structure localized above the environment. Robot recognition and pose tracking are accomplished using

[4] τ is the probability of changing the movement direction at each second and determines the randomness of the robot movement.

the ReactiVision software (Kaltenbrunner and Bencina, 2007).

Bibliography

Andreasson, H., Duckett, T., and Lilienthal, A. J. (2007). Mini-slam: Minimalistic visual slam in large-scale environments based on a new interpretation of image similarity. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 4096–4101.

Anon (1999). Intelligent flight control: Advanced concept program, final report. Technical report, The Boeing Company.

Antonelo, E. A., Baerlvedt, A.-J., Rognvaldsson, T., and Figueiredo, M. (2006). Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 498– 505, Vancouver, Canada.

Antonelo, E. A., Figueiredo, M., Baerlvedt, A.-J., and Calvo, R. (2005). Intelligent autonomous navigation for mobile robots: spatial concept acquisition and object discrimination. In *Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 553–557, Helsinki, Finland.

Antonelo, E. A., Schrauwen, B., and Stroobandt, D. (2008a). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 21:862–871.

Antonelo, E. A., Schrauwen, B., and Stroobandt, D. (2008b). Imitation learning of an intelligent navigation system for mobile robots

using reservoir computing. In *10th Brazilian Symposium on Neural Networks (SBRN)*.

Arkin, R. (1998). *Behavior-based robotics*. MIT Press.

Arleo, A., Smeraldi, F., and Gerstner, W. (2004). Cognitive navigation based on nonuniform gabor space sampling, unsupervised growing networks, and reinforcement learning. *IEEE Transactions on Neural Networks*, 15(3):639–652.

Atiya, A. F. and Parlos, A. G. (2000). New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11:697.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localisation and mapping (SLAM): Part ii state of the art. *Robotics and Automation Magazine*, pages 108–117.

Bakker, B. (2002). Reinforcement learning with long short-term memory. In *Advances in Neural Information Processing Systems*, volume 2, pages 1475–1482. MIT.

Bengio Y., Simard P., F. P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5:157–166.

Berkes, P. and Wiskott, L. (2005). Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5:579–602.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.

Braitenberg, V. (1984). *Vehicles: Experiments in synthetic psychology*. MIT Press.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Brooks, R. (1991). New approaches to robotics. *Science*, 253:1227–1232.

Brunel, N. and Trullier, O. (1998). Plasticity of directional place fields in a model of rodent ca3. *Hippocampus*, 8:651–665.

Buesing, 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.

Buonomano, D. and Maass, W. (2009). State-dependent computations: Spatiotemporal processing in cortical networks. *Nature Reviews Neuroscience*, 10(2):113–125.

Burgess, N., Barry, C., and O'Keefe, J. (2007). An oscillatory interference model of grid cell firing. *Hippocampus*, 17(9):801–812.

Bush, K. (2008). *An Echo State Model of non-Markovian Reinforcement Learning*. PhD thesis, Colorado State University, Fort Collins, CO.

Caprari, G., Arras, K., and Siegwart, R. (2001). Robot Navigation in Centimeter Range Labyrinths. In *5th International Heinz Nixdorf Symposium, Autonomous Minirobots for Research and Edutainment (AMiRE'01)*, Paderborn, Germany.

Chavarriaga, R., Str$\surd\partial$sslin, T., Sheynikhovich, D., and Gerstner, W. (2005). A computational model of parallel navigation systems in rodents. *Neuroinformatics*, 3:223–241.

Dominey, P. (1995). Complex sensory-motor sequence learning based on recurrent state representation and reinforcement learning. *Biological Cybernetics*, 73(3):265–274.

Doya, K. (1992). Bifurcations in the learning of recurrent neural networks. In *Proceedings of IEEE Int. Symp. on Circuits and Systems*, volume 6, pages 2777–2780.

Doya, K. (2002). Recurrent neural networks: Supervised learning. In Arbib, M., editor, *The Handbook of Brain Theory and Neural Networks, Second Edition*. MIT Press.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification - Second Edition*. John Wiley and Sons, Inc.

Eichenbaum, H., Dudchenko, P., Wood, E., Shapiro, M., and H, H. T. (1999). The hippocampus, memory, review and place cells: Is it spatial memory or a memory space? *Neuron*, 23:209–226.

Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, 6:503–556.

Filliat, D. and Meyer, J.-A. (2003). Map-based navigation in mobile robots:: I. a review of localization strategies. *Cognitive Systems Research*, 4(4):243 – 282.

Floreano, D. and Mondada, F. (1996). Evolution of homing navigation in a real mobile robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(3):396 –407.

Foldiak, P. and Endres, D. (2008). Sparse coding. *Scholarpedia*, 3(1):2984.

Forster, A., Graves, A., and Schmidhuber, J. (2007). RNN-based learning of compact maps for efficient robot localization. In *Proceedings of ESANN*.

Frank, L. M., Brown, E. N., and Wilson, M. (2000). Trajectory encoding in the hippocampus and entorhinal cortex. *Neuron*, 27(1):169 – 178.

Franzius, M., Sprekeler, H., and Wiskott, L. (2007a). Slowness and sparseness lead to place, head-direction, and spatial-view cells. *PLoS Computational Biology*, 3(8):1605–1622.

Franzius, M., Vollgraf, R., and Wiskott, L. (2007b). From grids to places. *Journal of Computational Neuroscience*, 22:297–299. 10.1007/s10827-006-0013-7.

Germund and Hesslow (2002). Conscious thought as simulation of behaviour and perception. *Trends in Cognitive Sciences*, 6(6):242–247.

Giralt, G., Chatila, R., and Vaisset, M. (1984). An integrated navigation and motion control system for autonomous multisensory

mobile robots. In *First International Symposium on Robotics Research*, pages 191–214. ed. M Brady and R. Paul.

Harnad, S. (2001). Grounding symbols in the analog world with neural nets - a hybrid model. *Psychology*, 12:12–78.

Harnad, S. (2003). The symbol grounding problem. In *Encyclopedia of Cognitive Science*. London: Nature Publishing Group/Macmillan.

Hashimoto, W. (2003). Quadratic forms in natural images. *Network: Comput. Neural Syst.*, 14:765–788.

Hasselmo, M. E. (2008). Grid cell mechanisms and function: Contributions of entorhinal persistent spiking and phase resetting. *Hippocampus*, 18(12):1213–1229.

Haykin, S. (1999). *Neural Networks: a comprehensive foundation (Second edition)*. Prentice Hall.

Hebb, D. O. (1949). *The Organization of Behavior*. New York: Wiley & Sons.

Hertzberg, J., Jaeger, H., and Schönherr, F. (2002). Learning to ground fact symbols in behavior-based robots. In *Proceedings of the 15th European Conference on Artificial Intelligence*, pages 708–712.

Heyes, C. M. (1994). Social learning in animals: categories and mechanisms. *Biological Reviews*, 69(2):207–231.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science USA*, 79(8):2554–2558.

Hyvärinen, A. and Oja, E. (1998). Independent component analysis by general nonlinear hebbian-like learning rules. *Signal Processing*, 64(3):301 – 313.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13:411–430.

Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for Information Technology.

Jaeger, H. (2002a). Short term memory in echo state networks. Technical Report GMD Report 152, German National Research Center for Information Technology.

Jaeger, H. (2002b). Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology.

Jaeger, H. and Haas, H. (2004). Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless telecommunication. *Science*, 308:78–80.

Jaeger, H., Lukosevicius, M., and Popovici, D. (2007). Optimization and applications of echo state networks with leaky integrator neurons. *Neural Networks*, 20:335–352.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546.

Kaltenbrunner, M. and Bencina, R. (2007). reacTIVision: a computer-vision framework for table-based tangible interaction. In *TEI '07: Proceedings of the 1st international conference on Tangible and embedded interaction*, pages 69–74, New York, NY, USA. ACM.

Kim, D. (2004). Evolving internal memory for T-maze tasks in noisy environments. *Connection Science*, 16(3):183–210.

Klampfl, S. and Maass, W. (2009). Replacing supervised classification learning by slow feature analysis in spiking neural networks. In *Proc. of NIPS 2009, Advances in Neural Information Processing Systems*, volume 22, pages 988–996. MIT Press.

Kuznetsov, Y. A. (1998). *Elements of Applied Bifurcation Theory*. Springer.

Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149.

LeCun, Y., Jackel, L. D., Boser, B., Denker, J. S., Graf, H. P., Guyon, I., Henderson, D., Howard, R. E., and Hubbard, W. (1989). Handwritten digit recognition: Applications of neural net chips and automatic learning. *IEEE Communication*, pages 41–46.

Li, N. and DiCarlo, J. J. (2008). Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science*, 321(5895):1502–1507.

Linaker, F. and Jacobsson, H. (2001). Mobile robot learning of delayed response tasks through event extraction: A solution to the road sign problem and beyond. In *Proc. IJCAI'2001*, pages 777–782.

Maass, W., Joshi, P., and Sontag, E. (2006). Principles of real-time computing with feedback applied to cortical microcircuit models. In *Advances in Neural Information Processing Systems*, volume 18.

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.

Mataric, M. (2001). Learning in behavior-based multi-robot systems: Policies, models, and other agents. *Cognitive Systems Research*, 2(1):81–93.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Michel, O. (2004). Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42.

Milford, M. (2008). *Robot Navigation from Nature*. Springer Tracts in Advanced Robotics.

Mondada, F. (2007). E-puck education robot. http://www.e-puck.org/.

Moravec, H. (1977). Towards automatic visual obstacle avoidance. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, page 584.

Moravec, H. (1983). The stanford cart and the cmu rover. *Proceedings of the IEEE*, 71(7):872–884.

Moser, E. I., Kropff, E., and Moser, M.-B. (2008). Place cells, grid cells and the brain's spatial representation system. *Annual Reviews of Neuroscience*, 31:69–89.

Nilsson, N. J. (1984). Shakey the robot. Technical Report tech. note 323, SRI AI Center.

O'Keefe, J. (1976). Place units in the hippocampus of the freely moving rat. *Experimental Neurology*, 51(1):78 – 109.

O'Keefe, J. and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34:171–175.

Ozturk, M. C., Xu, D., and Principe, J. C. (2006). Analysis and design of echo state networks. *Neural Computation*, 19:111–138.

Pavlov, I. P. (1927). *Conditioned Reflexes*. Oxford University Press.

Riedmiller, M. (2005). Neural fitted q iteration - first experiences with a data efficient neural reinforcement learning methods. In *In 16th European Conference on Machine Learning*, pages 317–328. Springer.

Rizzolatti, G. and Craighero, L. (2004). The mirror-neuron system. *Annual Review of Neuroscience*, 27:169–192.

Rosen, R. (1985). *Anticipatory Systems*. Pergamon Press.

Rosenstein, M. and Cohen, P. R. (1999). Concepts from time series. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 739–745.

Rumelhart, D., Hinton, G., and Williams, R. (1986). *Learning internal representations by error propagation*. MIT Press, Cambridge, MA.

Rylatt, R. M. and Czarnecki, C. A. (2000). Embedding connectionist autonomous agents in time: The 'road sign problem'. *Neural Processing Letters*, 12:145–158.

Schönherr, K., Cistelecan, M., Hertzberg, J., and Christaller, T. (2001). Extracting situation facts from activation value histories in behavior-based robots. In *KI-2001: Advances in Artificial Intelligence (Joint German/Austrian Conference on AI, Proceedings),*, pages 305–319. Springer (LNAI 2174).

Schrauwen, B., Busing, L., and Legenstein, R. (2008). On Computational Power and the Order-Chaos Phase Transition in Reservoir Computing. In *Proceedings of NIPS*.

Schrauwen, B., Defour, J., Verstraeten, D., and Van Campenhout, J. (2007). The introduction of time-scales in reservoir computing, applied to isolated digits recognition. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*.

Siegwart, R. and Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. Bradford Book.

Simard, P., Steinkraus, D., and Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 958–963.

Skinner, B. F. (1953). *Science And Human Behavior*. New York.

Solomon, J. H. and Hartmann, M. J. (2006). Sensing features with robotic whiskers. *Nature*, 443:525.

Steels, L. (1993). The artificial life roots of artificial intelligence. *Artificial Life*, 1:75–110.

Steil, J. J. (2004). Backpropagation-Decorrelation: Online recurrent learning with O(N) complexity. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 1, pages 843–848.

Ster, B., Dobnikar, A., et al. (1996). Neural networks in medical diagnosis: Comparison with other methods. In *Proceedings of the International Conference EANN*, volume 96, pages 427–430.

Stroesslin, T., Sheynikhovich, D., Chavarriaga, R., and Gerstner, W. (2005). Robust self-localisation and navigation based on hippocampal place cells. *Neural Networks*, 18(9):1125–1140.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S. and Barto, A. G. (2000). *Reinforcement Learning: An Introduction*. MIT Press.

Szita, I., Gyenes, V., and Loerincz, A. (2006). Reinforcement learning with echo state networks. In Kollias, S., Stafylopatis, A., Duch, W., and Oja, E., editors, *Artificial Neural Networks - ICANN 2006*, volume 4131 of *Lecture Notes in Computer Science*, pages 830–839. Springer Berlin / Heidelberg.

Tani, J. (1996). Model-based learning for mobile robot navigation from the dynamical systems perspective. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(3):421–436.

Tani, J. (2003). Learning to generate articulated behavior through the bottom-up and the top-down interaction processes. *Neural Networks*, 16:11–23.

Tani, J. (2007). On the interactions between top-down anticipation and bottom-up regression. *Frontiers in Neurorobotics*, 1.

Tani, J. and Nolfi, S. (1999). Learning to perceive the world as articulated: An approach for hierarchical learning in sensory-motor systems. *Neural Networks*, 12:1131–1141.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press.

Trullier, O., Wiener, S. I., Berthoz, A., and Meyer, J.-A. (1997). Biologically-based artificial navigation systems: Review and prospects. *Progress in Neurobiology*, 51(5):483–544.

Tychonoff, A. and Arsenin, V. Y. (1977). *Solutions of Ill-Posed Problems*. Washington: Winston & Sons.

Ulbricht, C. (1996). Handling time-warped sequences with neural networks. In Maes P., e. a., editor, *From Animals to Animats 4: Proc. Fourth Int. Conf. on Simulation of Adaptive Behaviour*, pages 180–192. MIT Press.

Verschure, P., Krose, B., and Pfeifer, R. (1992). Distributed adaptive control: The self-organization of structured behavior. *Robotics and Autonomous Systems*, 9(3):181–196.

Verschure, P., Voegtlin, T., and Douglas, R. (2003). Environmentally mediated synergy between perception and behaviour in mobile robots. *Nature*, 425:620–624.

Verstraeten, D., Dambre, J., Dutoit, X., and Schrauwen, B. (2010). Memory versus non-linearity in reservoirs. In *Neural Networks, International Joint Conference, Proceedings*, page 8, New York, NY, USA. IEEE.

Verstraeten, D. and Schrauwen, B. (2009). On the quantification of dynamics in reservoir computing. In *ICANN '09: Proceedings of the 19th International Conference on Artificial Neural Networks*, volume 5768, pages 985–994. Springer-Verlag.

Verstraeten, D., Schrauwen, B., D'Haene, M., and Stroobandt, D. (2007). A unifying comparison of reservoir computing methods. *Neural Networks*, 20:391–403.

Vogt, P. (2001). Symbol grounding in communicative mobile robots. In Coradeschi, S. and Saffiotti, A., editors, *Anchoring Symbols to Sensor Data in Single and Multiple Robot Systems: Papers from the 2001 AAAI Fall Symposium*, pages 87–94. AAAI Press.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proc. IEEE*, 78(10):1550–1560.

Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

Wilson, M. (2002). Six views of embodied cognition. *Psychonomic Bulletin & Review*, 9:625–636. 10.3758/BF03196322.

Wiskott, L. and Sejnowski, T. J. (2002). Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770.

wyffels, F., Schrauwen, B., Verstraeten, D., and Stroobandt, D. (2008). Band-pass reservoir computing. In Hou, Z. and Zhang, N., editors, *Proceedings of the International Joint Conference on Neural Networks*, pages 3203–3208, Hong Kong.

Wyss, R., König, P., and Verschure, P. F. M. J. (2006). A model of the ventral visual system based on temporal stability and local memory. *PLoS Biol*, 4(5):e120.

Yamashita, Y. and Tani, J. (2008). Emergence of functional hierarchy in a multiple timescale neural network model: A humanoid robot experiment. *PLoS Comput Biol*, 4(11):e1000220.

Yamazaki, T. and Tanaka, S. (2007). The cerebellum as a liquid state machine. *Neural Networks*, 20:290–297.

Zhang, K., Ginzburg, I., McNaughton, B. L., and Sejnowski, T. J. (1998). Interpreting neuronal population activity by reconstruction: unified framework with application to hippocampal place cells. *J Neurophysiol*, 79(2):1017–1044.

Ziemke, T. and Thieme, M. (2002). Neuromodulation of reactive sensorimotor mappings as a short-term memory mechanism in delayed response tasks. *Adaptive Behavior*, 10(3/4):185–199.

List of Publications

International journal publications

1. **Antonelo, E. A.** and Schrauwen, B. (2011). Learning slow features with reservoir computing for biologically-inspired robot localization. *Neural Networks*. in press.

2. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2008). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 21:862–871.

3. **Antonelo, E. A.**, Schrauwen, B., and Campenhout, J. V. (2007). Generative modeling of autonomous robots and their environments using reservoir computing. *Neural Processing Letters*, 26(3):233–249.

Full papers in proceedings of international conferences

1. **Antonelo, E. A.** and Schrauwen, B. (2010). Supervised learning of internal models for autonomous goal-oriented robot navigation using reservoir computing. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

2. **Antonelo, E. A.** and Schrauwen, B. (2009). Towards autonomous self-localization of small mobile robots using reservoir computing and slow feature analysis. In *Proceedings of*

the IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 3818–3823.

3. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2009). Unsupervised learning in reservoir computing: Modeling hippocampal place cells for small mobile robots. In *ICANN '09: Proceedings of the 19th International Conference on Artificial Neural Networks*, volume 5768, pages 747–756, Berlin, Heidelberg. Springer-Verlag.

4. Waegeman, T., **Antonelo, E. A.**, wyffels, F., and Schrauwen, B. (2009). Modular reservoir computing networks for imitation learning of multiple robot behaviors. In *Proc. of the IEEE Int. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*.

5. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2008). Modeling multiple autonomous robot behaviors and behavior switching with a single reservoir computing network. In *Proceedings of the IEEE International Conference on Man, Systems and Cybernetics (SMC)*.

6. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2008). Mobile robot control in the road sign problem using reservoir computing networks. In *Proceedings of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

7. **Antonelo, E. A.**, Schrauwen, B., Dutoit, X., Stroobandt, D., and Nuttin, M. (2007). Event detection and localization in mobile robot navigation using reservoir computing. In de et al., J. M., editor, *ICANN, Part II*, pages 660–669. Springer-Verlag.

8. **Antonelo, E. A.**, Baerlvedt, A.-J., Rognvaldsson, T., and Figueiredo, M. (2006). Modular neural network and classical reinforcement learning for autonomous robot navigation: Inhibiting undesirable behaviors. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 498– 505, Vancouver, Canada.

9. **Antonelo, E. A.**, Figueiredo, M., Baerlvedt, A.-J., and Calvo, R. (2005). Intelligent autonomous navigation for mobile robots:

spatial concept acquisition and object discrimination. In *Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, pages 553–557, Helsinki, Finland.

10. Calvo, R., Figueiredo, M., and **Antonelo, E. A.** (2005). Evolutionary fuzzy system for architecture control in a constructive neural network. In *Proceedings of the 6th IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*.

Full papers in proceedings of national conferences

1. **Antonelo, E. A.**, Depeweg, S., and Schrauwen, B. (2011). Learning navigation attractors for mobile robots with reinforcement learning and reservoir computing. In *Proceedings of the X Brazilian Congress on Computational Intelligence (CBIC)*. accepted.

2. **Antonelo, E. A.** and Schrauwen, B. (2009). On different learning approaches with echo state networks for localization of small mobile robots. In *Proceedings of the Brazilian Congress on Neural Networks (CBRN)*.

3. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2008). Imitation learning of an intelligent navigation system for mobile robots using reservoir computing. In *Proceedings of the 10th Brazilian Symposium on Neural Networks (SBRN)*.

4. **Antonelo, E. A.**, Schrauwen, B., and Stroobandt, D. (2007). Experiments with reservoir computing on the road sign problem. In *Proceedings of the VIII Brazilian Congress on Neural Networks (CBRN)*.

5. **Antonelo, E. A.** and Figueiredo, M. (2004). Sistemas autônomos inteligentes em navegação de robôs: Aquisição de conceitos espaciais e discriminação de objetos. In *Proceedings of the II Workshop on Intelligent Robotics (EnRI) - Brazilian Computing Society Congress*.