Neural Dynamics for Mobile Robot Adaptive Control

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Mohamed Oubbati

aus Algerien

Hauptberichter: Prof. Dr. Paul Levi
Mitberichter: Prof. Dr. Peter Göhner

Tag der mündlichen Prüfung: 27. Juni 2006

Institut für Parallele und Verteilte Systeme
der Universität Stuttgart

2006

Preface

This Thesis is submitted in partial fulfillment of the requirements for the degree of "Doktor der Naturwiessenschaften"(Dr. rer. nat.) at the Institute of Parallel and Distributed Systems of Stuttgart University. Prof. Dr. Paul Levi and Prof. Dr. Peter Göhner acted as referees.

Summary of the Thesis

In this thesis, we investigate how dynamics in recurrent neural networks can be used to solve some specific mobile robot problems.

We have designed a motion control approach based on a novel recurrent neural network. The advantage of this approach is that, no knowledge about the dynamic model is required, and no synaptic weight changing is needed in presence of time varying parameters. Furthermore, this approach allows a single fixed-weight network to act as a dynamic controller for several distinct robots.

To generate the robot behavior over time, we adopted the theory of neural fields. We designed a framework to navigate a robot to its goal in an unknown environment without any collisions with static or moving obstacles. In addition, we could optimize the target path through intermediate homebases. This framework has also produced a simple and elegant solution for the problem of moving multiple robots in formation. The objective was to acquire a target, avoid obstacles and keep a geometric configuration at the same time.

We have obtained successful results, both on simulations and on real experimentations.

Index Terms: Autonomous mobile robots, recurrent neural networks, metalearning, adaptive control, adaptive identification, neural field, navigation, behavior-based control, robot formation.

Acknowledgment

Over the last three years, I have had the privilege to meet many people who contributed either directly to my scientific work or to my quality of life in general.

First of all, I would like to express my gratitude to my advisors Prof. Dr. Paul Levi and Dr. Michael Schanz for supporting me, and for giving me so much freedom to explore and discover solutions for robotics. Thanks go also to my colleagues of the RoboCup COPS-team, with whom I enjoyed to work.

I would also like to acknowledge the many people in Marburg, Bonn, Bremen, and Stuttgart who have shown great kindness and support, both academically and administratively over the years.

Many thanks to all my friends, who helped me in many ways.

I am very grateful to the German Academic Exchange Service (DAAD) for funding my research. During four years, I have enjoyed my scholarship... Thanks a lot.

Finally, I can find no words to express my sincere appreciation and gratitude to the German people for their hospitality, kindness, punctuality, and for their passion for science and work.

Thank you Germany ... Herzlichen Dank Deutschland.

Mohamed Oubbati
Stuttgart, June 2006

Zusammenfassung

Zielsetzung

Autonome mobile Systeme müssen in der Lage sein, sich in ihre Einsatzumgebung autonom zu bewegen und dabei zu wissen, auf welchen Wegen sie sicher an bestimmte Zielpunkte gelangen. Ziel dieser Arbeit ist die Entwicklung eines Ansatzes, der mittels recurrent neuronaler Netze genaue und sichere zielgerichtete Bewegungen ermöglicht und dabei die kinematischen und dynamischen Einschränkungen eines Roboters berücksichtigt. Hierbei stellt sich die Frage, wie die Dynamik dieser Netze für die Kontrolle des beobachtbaren Verhaltens des Roboters eingesetzt werden kann. Im Rahmen dieser Arbeit soll diese Frage theoretisch, numerisch sowie praktisch untersucht werden.

Aufbau der Arbeit

Die Popularität der neuronalen Netze liegt darin, dass sie eine sehr flexible Modellklasse beschreiben. Neuronale Netze, deren Konnektivitätsgraf keine geschlossenen Wege enthält, werden heute in vielen Anwendungen eingesetzt. Solche zyklenfreie (feedforward) Netze realisieren nichtlineare Abbildungen und sind mit einfachen Lernverfahren trainierbar, besitzen jedoch keine eigene Dynamik. Rekurrente Netze, die durch einen Konnektivitätsgraf mit geschlossenen Wegen gekennzeichnet sind, besitzen dagegen ein weit reichenderes Spektrum an Verhaltensmöglichkeiten, da sie eine echte Dynamik aufweisen. Sie sind jedoch in ihrem Verhalten schwerer zu beherrschen und wurden daher in Anwendungen bisher selten eingesetzt.
Als Grundlage der Arbeit, wird zu Beginn ein Konstruktives Lernverfahren für rekurrente neuronale Netze untersucht, welches nur die Ausgabeneuronen führenden Verbindungen modifiziert, um das Lernziel zu erreichen. Diese Untersuchung ist durch einen speziellen Typ eines rekurrenten Netzes motiviert (Echo State Network, ESN), das in dieser Arbeit vor allem

aus Interesse an seiner Fähigkeit einfacher Lernverfahren, näher untersucht und für die Robotiksteuerung angepasst. Zudem wird gezeigt, dass das ESN durch *metalearning* ohne Änderungen seiner Parameter ein adaptives Verhalten hat.

Darauf aufbauend wird zunächst das ESN zur Identifikation nichtlinearer dynamischer Systeme behandelt. Hierfür werden nichtlineare dynamische Systeme ausgewählt, die durch nichtlineare Differenzengleichungen mit zeitvarianten Faktoren beschrieben werden. Es wird gezeigt, dass das ESN im Zusammenhang mit *metalearning* sehr gute Identifikationsergebnisse liefert und eine effektivere adaptive Echtzeitidentifikation im Vergleich zu anderen bekannten Verfahren ermöglicht.

Zur Steuerung autonomer mobiler Roboter wird anhand des Robotersmodells eine adaptive Geschwindigkeitsregelung mittels eines ESN entworfen, simuliert, realisiert und auf einem realen Roboter getestet. In der Simulation wird ein Neurokontrol-System mit zwei Ebenen entwickelt. Die eine Ebene (dynamic-level) für die adaptive Kontrolle des dynamischen Modells, die auf dem ESN und metalearning basiert; die zweite Ebene (kinematic-level) für das kinematische Modell, die auf einem anderen ESN basiert. Um die Robustheit zu verbessern, wurde während des Trainings zusätzlich ein Geräusch hinzugefgt. Hierbei konnten positive Ergebnisse erzielt werden.

Auf dem realen Roboter wird ein ESN zu einem Pulsweitenmodulationsregler ausgebildet, der aufgrund der gegebenen aktuellen Geschwindigkeiten und der Sollgeschwindigkeiten die korrekten Parameter der Pulsweitenmodulation bestimmt. Ein Vorteil dieses Ansatzes ist die Anpassung der Radgeschwindigkeitsreglung ohne dass vorher Kenntnisse über die dynamischen Eigenschaften des Roboters vorliegen müssen. Zudem kann sich ein einmal vortrainierter ESN-Regler bei Variation der Roboterphysik, z.B. bei Gewichtsveränderungen, ohne Änderungen seiner Parameter, auf die Änderungen relativ schnell einstellen. Diese Methode ermöglicht auch die gleichzeitige Geschwindigkeitsregelung drei unterschiedlicher Roboter mit einem einzigen vortrainierten ESN-Regler.

Zur Navigation werden dynamische Systeme verwendet, um Navigationsverhalten für einen oder mehrere autonome Roboter zu produzieren. Insbesondere soll ein biologisch motiviertes neuronales Feldmodell genutzt werden, um ein stabiles Verhalten eines Roboters zu erzeugen. Neuronale Felder sind skalare räumlich kontinuierliche Aktivitätsverteilungen, die typischerweise die Aktivität des Neokortex beschreiben. Die Felddynamik ist durch

nichtlineare Integral-Differentialgleichungen beschrieben, die sich folgendermaßen darstellen: Der Ort der Neuronen wird durch die kontinuierliche Variable x bezeichnet. Eine orts- und zeitabhängige Funktion (Aktivierung) $u(x,t)$ repräsentiert dann den Zustand der Dynamik an jedem Ort x zur Zeit t. Unter Annahme der Gleichgewichtslösung und externer Stimuli wird ein einziger Gipfel, im Folgenden "Peak" genannt, auf dem Feld erzeugt. In diesem Fall werden die Verhalten *Hindernisvermeidung* und *Zielanfahrt* durch das Feldmodell generiert. Zuerst werden alle möglichen Richtungseinstellungen des Roboters kodiert; dann gibt der Ort, an dem ein Peak generiert wurde, die Soll-Vorausrichtungen an. Eine vorgeschlagene Richtung wird solange beibehalten bis eine Änderung in der Umwelt auftritt. Hierdurch wird gezeigt, dass die stabilen Entscheidungen der neuronalen Felder, eine sichere Bewegung unter Vermeidung von statischen und dynamischen Hindernissen ermöglichen. Neben der Möglichkeit, Hindernisse zu vermeiden, wird das entwickelte lokale Navigationsystem angepasst, um auf dem Weg zum Zielpunkt zusätzlich *homebases* anzusteuern.
Darüberhinaus wird im Rahmen dieser Arbeit ein Navigationsystem zur koordinierten Steuerung eines Multirobotersystem entwickelt. Es wird eine Neuronale Felder Strategie vorgestellt, mit der es möglich ist, eine Gruppe von Robotern zu steuern, die ihren Zielpunkt erreichen, eine Formation beibehalten, und Kollisionen mit Hindernissen oder miteinander vermeiden.

Schlagwörter: Mobile Roboter, Autonome Navigation, Rekurrente Neuronale Netze, Adaptive Regelung, Adaptive Identifikation, Metalearning, Verhaltensbasierte Kontrolle, Neuronale Felder.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 What sort of thesis is this?

This thesis is a robotics thesis. We address issues of machine learning. We investigate how recurrent neural networks can offer efficient and practical solutions. We design adaptive systems. But, this thesis describes progress towards the ultimate goal of mobile robotics: "*move a robot safely and precisely in real environments*".

1.1.1 Tasks

Two main tasks are addressed in this work. The first task is designing an adaptive motion control system. The second is the behavior-based control of single and multiple mobile robots. Both of these tasks involve real-world constraints.

1.1.2 How these tasks are addressed?

The primary methods we used for solving our robotics problems are based on recurrent neural network (RNN) techniques. The ability of RNNs to instantiate arbitrary temporal dynamics allows us to design motion control, includ-

ing also the dynamics part of the robot. Moreover, through metalearning, no synaptic weight changing is needed in presence of parameters variation. The concept of dynamic stability in neural fields is adopted as a framework for behavior-based control. A self-stabilized peak of activation is linked to on-line sensory information and decodes optimal behaviors.

1.2 Issues and Goals

This section describes more details of the tasks we address, and what questions we examine in each task.

1.2.1 Motion Control

Many authors have studied motion control of mobile robots in the last decade. At the beginning, the research effort was focused only on the kinematic level, assuming that there is perfect velocity tracking. Later on, the research has been conducted to design motion controllers, including also the dynamics part of the robot. Taking into account the specific robot dynamics is more realistic, because the assumption "perfect" velocity tracking does not hold in practice. Furthermore, during the motion, the robot parameters may change due to surface friction, additional load, energy supply, among others. Therefore, the control at the dynamic level is at least as important as the control at the kinematic level.
At present, PID controllers are widely used as velocity controllers for mobile robots. However, their ability to cope with nonlinearities and time varying parameters is known to be very poor. In recent years, renewed interest has been shown in the area of systems control using nonlinear control theory. Instead of using an approximate linear model, nonlinear models are used and nonlinear feedbacks are employed on the control loop. However, nonlinear controllers have a more complicated structure and are more difficult to find. Furthermore, exact knowledge about systems parameters values is almost unattainable in practical situations.

In this issue, it is desirable to develop a robust motion control, which has the following capabilities: i) ability to successfully handle errors and noise in sensor signals, ii) "*perfect*" velocity tracking, and iii) adaptation ability in presence of time varying parameters in the robot.

1.2.2 Behavior Generation

While moving, the robot must be able to collect knowledge about the environment, find its location, plan its path, and react to new situations. Approaches that have been developed for this problem can be divided into global and local methods. Global methods require the environment to be completely known and the terrain should be static, and they return a continuous free path. By contrast, local methods need only local information, and the robot plans its path in response to environmental changes. Due to their low computational costs, local methods are much more suitable for real-time applications where the environmental state changes continually.
Over the last decade, a new paradigm called *behavior-based robotics* has been established. Instead of the classical chain "Sense-Plan-Act", a behavior-based system is divided up into a set of *perception-action* units, called *behaviors*. Each behavior produces immediate reactions in order to achieve a specific task. One of the central design challenges of behavior-based systems is to find effective mechanisms for coordination between behaviors.
The so-called *Dynamical Systems Approach* presents a new framework that unifies both, design of behaviors and their coordination. One great advantage of this approach is that a continuous control signal can be assured at all times. While their suitability has been already proven to solve the problem of target-acquisition with obstacle avoidance in simplified real-world settings, more complex problems need to be tackled: i) target acquisition and avoiding moving obstacles, ii) multi-target acquisition, and iii) target acquisition, avoiding obstacles, and formation control of multiple mobile robots.

1.3 Contributions

This thesis describes a research work in which the problems cited above are attacked. It proceeds along two principal directions: recurrent neural networks, and mobile robot control.
In the first direction, we study the possibility that a readout neuron can learn to extract salient information from a high dimensional transient state of a large RNN. This ability reduces significantly the learning complexity of RNNs, and open new issues to benefit from their powerful capabilities. Echo State Network (ESN), developed recently, has the same philosophy. It is formed by a so-called "*Dynamic Reservoir*, which contains a large num-

ber of sparsely interconnected neurons with non-trainable weights. This idea leads to a simple off-line training algorithm where *only* the network-to-output connection weights have to be trained. However, the on-line version is computationally very expensive, and poor performances have been obtained in real-time adaptation. Therefore, one issue of this thesis was to explore the notion that fixed-weight RNNs need to change only their internal state to change their behavior policy. This is possible, since recurrent signal loops can store information by accumulation, and they act like weights in a conventional feedforward network. Through the concept of meta-learning, we could build an ESN, which has the ability to adapt without changing its weights. This approach was first tested in nonlinear dynamical system identification. The ESN identifier has the ability to recognize the system dynamics variations only through its inputs, and its own state, without changing any synaptic weight. An advantage of ESN is that no multi-streaming is needed, since its training algorithm uses all data for a single time, and does not suffer from the recency effect. A conference publication related to this work is:

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Meta-learning for Adaptive Identification of Non-linear Dynamical Systems. In: *Proceeding of the Joint 20th IEEE International Symposium on Intelligent Control and 13th Mediterranean Conference on Control and Automation*, Limassol, Cyprus, June 2005.

In motion control, we designed firstly a dynamic controller based on ESNs. The advantage of the control approach is that no knowledge about the robot model is required, since the controller is designed only by learning I/O data collected from the robot. This property is very useful in practical situations, where the exact knowledge about the robot parameters is almost unattainable. The proposed approach has been tested through simulations and experimentations on an Omnidirectional RoboCup Player available at the Robotics Lab of the University of Stuttgart. Results are summarized in two papers:

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Recurrent Neural Network for Wheeled Mobile Robot Control. In: *WSEAS Transaction on Systems*, 3:2460-2467, August 2004.

- Oubbati Mohamed, Schanz Michael, Buchheim Thorsten, and Levi Paul. Velocity Control of an Omnidirectional RoboCup Player with Recurrent Neural Networks. In: *Proceedings of the RoboCup International Symposium 2005*, July 18-19, 2005, Osaka,Japan.

To deal with robustness and time varying parameters, an adaptive neurocontrol system with two levels was proposed for the motion control of a nonholonomic mobile robot. In the first level, an ESN (called ESN_K) improves the robustness of a kinematic controller and generates linear and angular velocities, necessary to track a reference trajectory. In the second level, another network (called ESN_D) converts the desired velocities, provided by the first level, into a torque control. The advantage of the control approach is that, no knowledge about the dynamic model is required, and no synaptic weight changing is needed in presence of robots parameters variation. Furthermore, we demonstrated the ability of a robust single fixed weight ESN to act as a dynamic controller for several distinct wheeled mobile robots. The controller showed a reasonable balance between the variety of the reference velocity and the variety of the robots. Simulation and experimental results can be found in the following publications:

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Kinematic and Dynamic Adaptive Control of a Nonholonomic Mobile Robot using a RNN. In: *Proceedings of the 6th IEEE Symposium on Computational Intelligence in Robotics and Automation*, June 27-30, 2005, Helsinki, Finland.

- Oubbati Mohamed, Schanz Michael, and Levi Paul. A fixed weight RNN dynamic controller for multiple mobile robots. In: *Proceedings of the 24th IASTED International Conference on Modelling, Identification, and Control: MIC 2005*,February 16-18, 2005, Innsbruck, Austria.

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Fixed-weight RNN Adaptive Controller for an Omnidirectional Robot. In: *Proceedings of the 9th International Conference on Engineering Applications of Neural Networks (EANN05)*, August 24-26 2005, Lille, France.

- Oubbati Mohamed, Schanz Michael, and Levi Paul.Mobile Robot Motion using Neural Networks: An Overview. In: *19. Fachgespraech Autonome Mobile Systeme AMS 2005*, 8-9 December 2005, Stuttgart, Germany.

In behavior-based control, the concept of neural fields was used to generate the robot behavior over time. We developed a framework to navigate a mobile robot to its goal in an unknown environment without any collisions

with static or moving obstacles. In addition, their competitive dynamics was used to optimize the target path through intermediate home-bases. Furthermore, we could design an elegant solution for the problem of moving multiple robots in formation. The objective was to acquire a target, avoid obstacles and keep a geometric configuration at the same time. Several formations for a team of three robots were considered.
In all these developments, the speed control was fully integrated, relative to each behavior. In earlier studies, this quantity was usually set to a constant value. Two papers have been published:

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Neural Fields for Behavior-based Control of Mobile Robots. In: *8th International IFAC Symposium on Robot Control(SYROCO 2006)*, September 6-8 2006, Bologna, Italy.

- Oubbati Mohamed, Schanz Michael, and Levi Paul. Neural Fields for Controlling Formation of Multiple Robots. In: *3rd IEEE Conference On Intelligent Systems*. 4-6 September 2006, London, UK.

1.4 Outline of the Thesis

The thesis is organized as follows:

Chapter 1. Introduction

This chapter gives an overview of the thesis, its motivations, and main contributions. It also gives the list of papers published during the completion of the thesis.

Chapter 2. Related Work

This chapter reviews relevant related work. It gives an overview of different approaches concerning mobile robot control. Special attention is paid to motion control with neural networks, and behavior-based control.

Chapter 3. Recurrent Neural Networks

This chapter gives some background on recurrent neural networks. It covers the currently most important learning methods BPTT, RTRL, and EKF. It

focuses on random RNNs, and shows, that it is possible to transform transient state of a large RNN into a stable readout unit. At the end of the chapter, the architecture of the Echo State Network is formally described.

Chapter 4. Meta-Learning

The concept of meta-learning is discussed in chapter 4, focusing on neural networks, which have the ability to adapt without changing explicit weights. To illustrate its use, a fixed-weight ESN is built up for adaptive identification of non-linear dynamical systems with time varying parameters.

Chapter 5. Control of Nonholonomic Robots with ESNs

This chapter is concerned with the tracking control at the kinematic and dynamic level of nonholonomic mobile robots. After the description of the robot model, an adaptive neurocontrol system with two levels is designed for the motion control. It will be also shown how a single fixed-weight ESN is able to act as a dynamic controller for several (here 3) distinct robots.

Chapter 6. Control of an Omnidirectional Robot with ESNs

Real implementation of ESNs control approach is presented in this chapter. We firstly describe the hardware of the robot. After that, we will show how the ESN controller is designed only by learning I/O data collected from the robot. Furthermore, the ESN is trained using meta-learning to act as an adaptive velocity-tracking controller, in presence of global mass variation. Experimental results are presented.

Chapter 7. Neural Fields for Behavior Generation

This chapter presents firstly an introduction of instantiated dynamical systems approach. Then, we present the navigation system, which is entirely based on the concept of neural fields. We also present the extension of this approach to solve the problem of controlling formation of multi-robots.

Chapter 8. Neural Fields for Behavior-Based Control of an Omnidirectional RoboCup Player

In this chapter the behavior-based control with neural fields is implemented on a real omnidirectional robot. We will show how neural fields can solve some real-time navigation problems in some RoboCup scenarios. Experimental results are presented.

Chapter 9. Conclusion

This chapter draws conclusions, reviews the contributions, and point out some open problems and future lines of research.

Chapter 2

Related Work

This thesis has proceeded along two principal directions: behavior-based control and motion control. The objective is to provide the mobile robot the capability to solve several problems to successfully perform navigation in an environment that is unknown. In this chapter we review relevant related work.

2.1 Control Architectures

While there are infinitely many possible robot control programs, there is a small set of fundamental approaches. The main difference between them relies on whether they are more deliberative or more reactive. In this section the three main methodology are presented: purely deliberative, purely reactive, and hybrid architectures.

2.1.1 Deliberative Control

Since the first robots began to be built, the usual approach was to decompose the control problem into a series of functional units as illustrated in figure (2.1). First, the perception module gets the environment information

delivered by the sensors, which will be transformed to a world-model by the modelling module. The information in the world model is then used by the planning module to produce the appropriate actions to the actuators. This architecture is also called *sense-plan-act* (SPA) architectures.

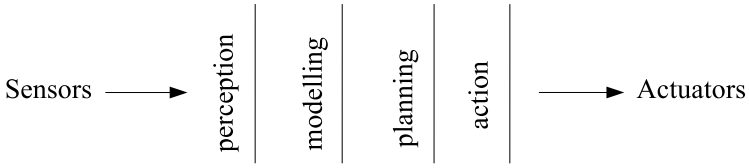


Figure 2.1: SPA Architecture.

This strategy works well when the environment is known and pre-defined (e.g. factories with marked paths). However, noisy and continually changing environments require frequent re-planning, which may be prohibitive for complex tasks. Another drawback of such strategy is the lack of robustness. Since information is processed in a serial way; a failure in one module causes a complete breakdown of the system.

2.1.2 Reactive Control

Purely reactive approaches are proposed to achieve real-time performances. Reactive systems maintain no internal world-model and perform no search for the optimum path. They typically apply a simple mapping function (i.e. pre-programmed condition-actions pairs) between sensors and actions to response rapidly to the environment feedback [1]. However, they are incapable of performing complex tasks, which needed to be planned.

2.1.3 Hybrid Control

Hybrid approaches attempt a compromise between purely deliberative and purely reactive approaches. They contain both reactive and deliberative components. The reactive component deals with the robot's immediate needs, such as avoiding obstacles, while the deliberative component uses the internal world-model to reason about what actions to take on a longer time-scale. The big challenge of this methodology is to construct an intermediate component, which controls the interaction between the two components. Many

approaches have been proposed: The *Three Layered Architectures* [2], *Servo-Subsumption-Symbolic* [3], and others [4].

2.2 Behavior-based Control

Behavior-based systems are inspired from biology, and try to model how animals deal with their environments. Typically, animals dont have a detailed model of objects they interact with, and minimal information and a combination of several behaviors are sufficient for them to behave efficiently in their environment [5][6]. This suggests that building a competent robot does not necessarily need a huge amount of representations.

The most prominent behavior-based control architecture is the *Subsumption* Architecture, first proposed by Rodney Brooks in 1986 [7]. Instead of a single chain Sense-Plan-Act, a behavior-based system is divided up into a set of *Perception-Action* units, called behaviors (see figure (2.2)). Based on selective information, each behavior produces immediate reactions in order to achieve a specific task. These behaviors are typically associated with some tasks, such as target acquisition, obstacle avoidance, wall following, etc. The advantage is that each behavior has to be competent only on the function delegated to its layer of control. Such "task-specializing" modules are much easier to built and run quite rapidly than serial functional units. Furthermore, this architecture can be expanded by simply adding more behaviors to the existing control system. A nice review of this approach can be found in [8].

While behavior-based systems embody some reactive components, their computation is not limited to a simple mapping function between sensors and actions. They can store representations in a distribution fashion through different behaviors [9][10], while reactive systems cannot do so.

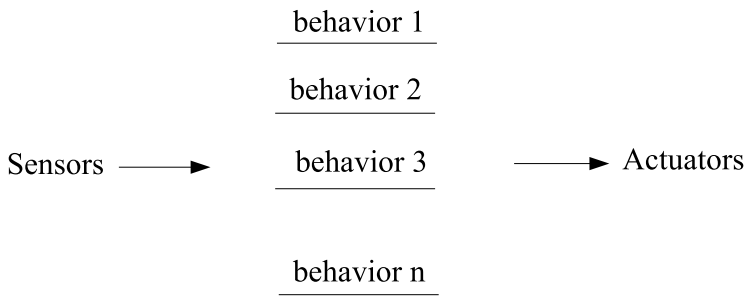


Figure 2.2: Behavior-based Control

2.2.1 Behavior Coordination

One of the central design challenges of behavior-based systems is the formulation of effective mechanisms for coordination between behaviors. At each time, it must be decided which behavior should be active. This is known as the behavior coordination problem (also referred to as the action selection problem). The existing coordination approaches can be divided into two classes: arbitration mechanisms and fusion mechanisms. More detailed information can be found in [11].

2.2.2 Arbitration Mechanisms

Arbitration Mechanisms are divided into: *priority*, *state*, and *winner-takes-all* based mechanisms. In *priority* based mechanism, a behavior is selected based on a priory assigned priorities. Behaviors with higher priorities are allowed to take the control of the robot. The *Subsumption* architecture [7] is an example using this type of selection. *state* based mechanisms select the appropriate behavior corresponding to the robot's states. Each state corresponds to one behavior. Upon detection of a new event, a transition is made to a new state; thus, a new behavior is activated. Discrete Event Systems [12], Reinforcement Learning [13], and Bayesian Decision Analysis [14] are methods where this mechanism is applied. Finally, in *winner-takes-all* mechanisms, the behaviors compete according to the situation until one behavior wins and take the control of the robot [15].

2.2.3 Fusion Mechanisms

Fusion mechanisms combine several behaviors at the same time, to form a control action that represents their consensus. Usually, these behaviors have different, and sometimes incompatible objectives. Many approaches have been proposed to select a motor command from multiple objectives. They can be roughly classified into *Voting*, *Fuzzy*, and *Superposition* approaches. In voting schemes [16], each behavior votes "for" or "against" possible robot actions. At each situation, the action, which received the maximum of votes, will be selected. Fuzzy approaches are similar to voting approaches. However, Instead of voting, behaviors are encoded by fuzzy rule bases that map perceptions to actions. After combination of these behaviors, the resulting membership function is *defuzzyfied* to compute the motor command [17][18][19]. *Superposition* techniques fuse different behaviors by linear combinations. The most popular superposition approach is the *Potential Fields*, proposed by Khatib [20]. The idea is to consider that the robot moves under influences of an artificial potential field. The target applies an attractive force to the robot, while obstacles exert repulsive forces onto the robot. The sum of all forces determines the subsequent direction of the movement.

2.3 Dynamical Systems Approach

The so-called Dynamic Approach invented by Schöner in 1995 [21] presents a fundamentally different approach to behavior selection. It uses the theory of nonlinear dynamical systems to provide a framework that unifies both the design of behaviors and their coordination. This theory has proven to be an elegant and easy to generate robot behavior [22, 23, 24, 25]. It is based on differential equations for so-called *behavioral variables* the solution of which generates the robot's behavior.

2.3.1 Behavioral Variables

To design a behavior, we need to parameterize it by a variable (scalar or vector) $X \in R^N$. This variable defines quantitatively the state of the system projected on the corresponding behavioral dimension. Variables that fulfill this characterization are called *behavioral variables*. Generally, a behavior is related to the task at hand. This means, the behavioral variables must be

chosen in such a way that the task can be accomplished. In a navigation task, for example, behavior variables are the heading direction φ and the forward velocity v of an autonomous mobile robot.

2.3.2 Behavioral dynamics

The next step towards generating behavior is to evolve in time the solutions of a dynamical system that governs the behavioral variables. A requirement is that these solutions must at all time be or in close to a stable attractor state of the dynamics. The attractors are defined by the task to solve.

Dynamical systems are formulated as differential equations:

$$\dot{x} = f(x, p, t) \tag{2.1}$$

where $\dot{x} = \frac{dx}{dt}$ denotes the time derivative of a variable x, and f is a function of x, of a set of parameters p, and of time t. This function is designed in such a way that the set of points x_b defining a task constraint are *fixed points* of the dynamical system:

$$\frac{dx}{dt}|_{x=x_b} = f(x, p, t) = 0 \tag{2.2}$$

If x_b is a desired value of the behavior (for example the desired direction) than the fixed point must be an attractor. On the other hand, if x_b is an undesired one (for example the direction of an obstacle), the fixed point has to be a repellor.

An initial perturbation from an attractor needs a certain time τ to decay. In this sense, the strength of the attractor can be defined by $\lambda = \tau^{-1}$. This can be expressed by the slope of the dynamics at the fixed point.

$$\lambda_b = -\frac{\partial f(x, p, t)}{\partial x}|_{x=x_b} \tag{2.3}$$

A negative slope characterizes an attractor of the dynamics, a positive slope a repellor. A null variation means that the system is in a quasi-stable state.

2.3.3 Dynamical systems for mobile robot navigation

A typical mission of a mobile robot is to reach a target while avoiding obstacles. Following the methodology of the dynamical systems approach, the

behavioral variables x have to be defined first. Because directions and restrictions on speeds usually define navigation constraints, behavioral variables can be chosen as the heading direction φ and the forward velocity v of the robot:

$$x = \begin{pmatrix} \varphi \\ v \end{pmatrix} \tag{2.4}$$

Usually the dynamics of φ and v are not depending on each other:

$$\begin{pmatrix} \dot{\varphi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} f_\varphi(\varphi, p, t) \\ f_v(v, p, t) \end{pmatrix} \tag{2.5}$$

These systems are chosen in the simplest mathematical form possible, such that the desired behavior is achieved. If we assume that the forward velocity v is kept constant, then dynamics for φ consists of contributions of two dynamics each of which contributes with a strength λ; a dynamics of the *target-acquisition* and of *obstacle-avoidance*:

$$\dot{\varphi} = \lambda_{tar} f_{tar} + \lambda_{obs} f_{obs} \tag{2.6}$$

Target Acquisition

The behavior *target-acquisition* is expected to align the robot's heading with the target direction φ_{tar}. This angle specifies the position of an attractor in the movement generation dynamics. The simplest form that meets this criterion is given by [22, 25, 26, 27]:

$$f_{tar} = sin(\varphi_{tar} - \varphi) \tag{2.7}$$

A plot of the dynamical system in phase space can be seen in Figure(2.3). The intersection with the φ-axis defines the fixed point ($\varphi = 0$). Since the slope of f_{tar} at this point is negative, the fixed point is an attractor.

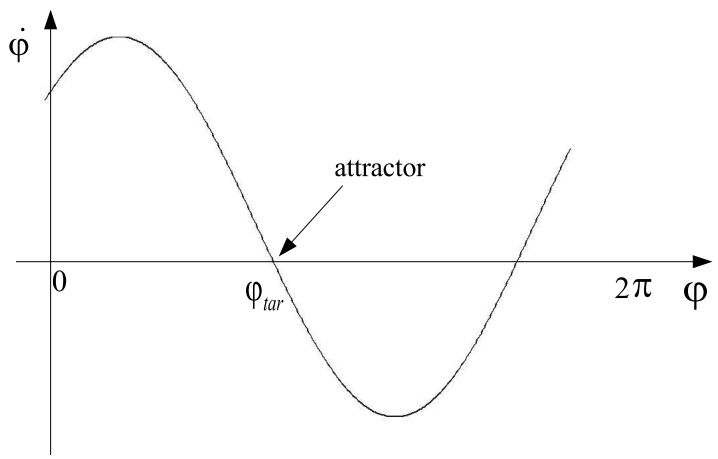


Figure 2.3: The dynamics of heading direction φ for *target-acquisition*. An attractor is generated at the direction φ_{tar}.

Obstacle Avoidance

The behavior *obstacle-avoidance* is expected to turn the robot away from the direction of obstacles. Thus, the dynamics should create a repellor along the obstacle direction . The dynamics is constructed for example as [26, 23, 24]:

$$f_{obs} = (\varphi_{obs} - \varphi) \exp\left(-\frac{(\varphi_{obs} - \varphi)^2}{\sigma}\right) \tag{2.8}$$

where σ is a positive constant that defines the angular range of the repellor. A plot of this dynamical system in phase space can be seen in Figure(2.4).

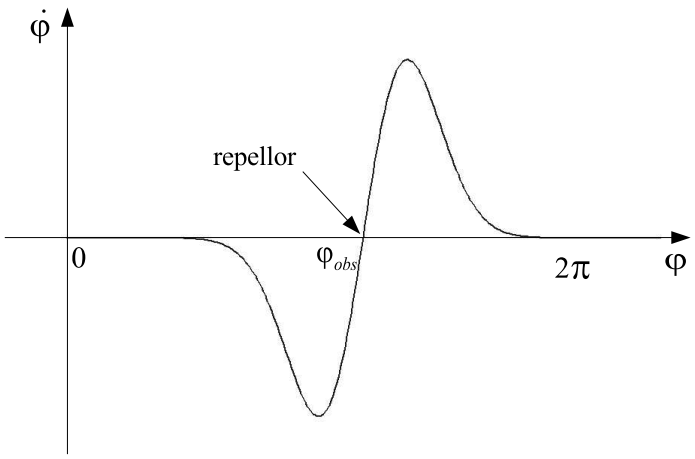


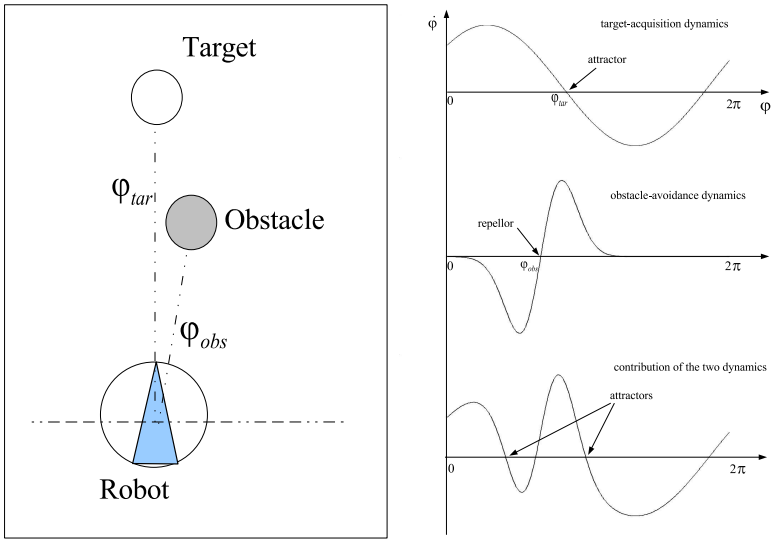
Figure 2.4: The dynamics of heading direction φ for *obstacle-avoidance*. A repellor is generated at the direction φ_{obs}.

Figure (2.5) shows a simple interaction between the two behaviors target-acquisition (attractor) and obstacle-avoidance (repellor). The dynamics of the robot heading angle is depicted by figure (2.5.b). From equation (2.6), this dynamics is the result of contributions of the target and obstacle functions (Last graph in figure(2.5).b). The presence of two final attractors, indicated by the two arrows, show the two possible ways to get to the target. Other more complex examples solved by dynamical systems can be found in [27].

In case of multiple N obstacles, the resulting force f_{obs} is computed by adding the contributions of individual obstacles.

$$f_{obs} = \sum_{i=1}^{N} \lambda_i(\varphi_i - \varphi) \exp\left(-\frac{(\varphi_i - \varphi)^2}{\sigma_i}\right) \tag{2.9}$$

The strength λ_i of each repellor depends on the distance of the obstacles. The robot heading direction is repelled strongly from near obstacles and weakly from far obstacles.



(a) (b)

Figure 2.5: Attractor and Repellor interaction

2.4 Motion Control

2.4.1 Background

Motion control may be divided into three basic points: tracking a reference trajectory, following a path, and point stabilization. The three basic navigation problems are presented briefly as follows:

Tracking

The trajectory-tracking problem is posed as follows. Assume that our robot has the posture: $P = (x, y, \theta)^T$, and that the reference robot (to be followed) has the posture:$P_r = (x_r, y_r, \theta_r)^T$. The objective is to find control laws for the linear and angular velocities (v, w) of our robot such as:
$\lim_{t\to\infty} |x(t) - x_r(t)| = 0$, $\lim_{t\to\infty} |y(t) - y_r(t)| = 0$, $\lim_{t\to\infty} |\theta(t) - \theta_r(t)| = 0$.

Path Following

Given a path in the plane ξ, and assuming that e_θ and e_{xy} are the orientation error and the distance between a reference point in the mobile robot and the

path ξ , respectively. The objective is to find a velocity control law, such that $\lim_{t\to\infty} |e_\theta| = 0$ and $\lim_{t\to\infty} |e_{xy}| = 0$.

Point Stabilization

Assume that the robot is on the initial pose P. Given an arbitrary configuration pose P_r . The objective of Point Stabilization is to find a velocity control law such that $\lim_{t\to\infty}(P - P_r) = 0$.

Motion control of mobile robots has been studied by many authors in the last decade, since they are increasingly used in wide range of applications. At the beginning, the research effort was focused only on the kinematic model, assuming that there is *perfect* velocity tracking [28]. The main objective was to find suitable velocity control inputs, which stabilize the kinematic closed-loop control. Later on, the research has been conducted to design motion controllers, including also the dynamics of the robot. However, when the dynamics part is considered, exact knowledge about the parameters values of the mobile robot is almost unattainable in practical situations. If we consider that during the robot motion, these parameters may change due to surface friction, additional load, among others, the problem becomes more complicated. Furthermore, the control at the kinematic level may be instable if there is errors control at the dynamic level. Therefore, the control at the dynamic level is at least as important as the kinematic-level control. At present PID controllers are still widely used in motor control of mobile robots and in industrial control systems in general. However, its ability to cope with some complex process properties such as non-linearities, and time-varying parameters is known to be very poor. Recently, some investigations have been conducted to design non-linear dynamic controllers. Instead of using approximate linear models as in the design of conventional linear controllers, non-linear models are used and non-linear feedback are employed on the control loop. Using non-linear controllers, system stability can be improved significantly; a few results are available in [29][30]. However, non-linear controllers have a more complicated structure, and are more difficult to find and to implement.

2.4.2 Motion Control with Neural Networks

In the past few years, Neural Networks (NNs) have been investigated extensively providing complementary/new solutions for identification and control of non-linear systems. Their ability to handle complex input-output mapping, without detailed analytical model, and robustness for noise environment make them an ideal choice for real implementations. Furthermore, neural networks deal with cognitive tasks such as learning, generalization, and adaptation, which constitute the principal navigation problems. In [31] a model predictive control (MPC) based on a neural network is proposed. Initially, a PID controller controls the robot to track a path in order to produce training data for the NN. Once the network is trained, the PID is replaced by the MPC controller, and an on-line learning is used during the path tracking. In [32] a tracking controller using backstepping technique is proposed and two MLPs are used as inverse controllers for the two DC motors. Experimental results on a low-quality mobile robot show high robustness against noise signals with an improvment in the path tracking compared with the PID controller. In [33] a kinematic controller based on back-stepping method is used for steering, and using Lyapunov stability, a velocity controller is developped for the dynamic model. In the control law, the nonlinear dynamical model of the robot is approximated by a NN. A comparison is made between three controllers: i) a controller that assumes "perfect velocity tracking"; ii) a controller that assumes a complete knowledge of the robot dynamics; and iii) a controller based on NN to model the robot dynamics. The performance of the control system was clearly improved by the NN Backstepping controller compared with the two others. Furthermore, no prior information about the robot dynamics was required. However, in that paper it was not clear how to train the NN and how to use the proposed approach in practice. The same authors in [34] used the same approach for point stabilization (parking) of a nonholonomic mobile robot.
Fusion of NNs with fuzzy logic is adopted by many authors for navigation control. In neuro-fuzzy control, usually the NN tunes the fuzzy control rules and membership functions [35][36].

Chapter 3

Recurrent Neural Networks

This chapter presents an overview on recent developments on learning with recurrent neural networks (RNNs), covering the currently most important ones i.e. BackPropagation Through Time (BPTT), Real-Time Recurrent Learning (RTRL) and the Extended Kalman Filter (EKF). Then it focuses on random RNNs, where it is shown the possibility to transform transient state of a large RNN into a stable readout unit, and how to use it to extract information during transient dynamics. Finally, the architecture of the Echo State Network (ESN), which will be used in the remainder of the thesis, is formally described.

3.1 Introduction

Feedforward networks have been successfully used to solve problems that require the computation of a static function, i.e a function whose output depends only upon the current input, and not on any previous inputs. In the real world however, one encounters many problems, which cannot be solved by learning a static function because the function being computed changes with each input received. It is clear from the architecture of feedforward neural networks that past inputs have no way of influencing the processing of

future inputs. This situation can be rectified by the introduction of feedback (recurrent) connections in the network. Recurrent neural networks have an internal state, which is essential for many domains where time plays a role, such as telecommunication, adaptive identification and control, time series prediction, or robotics to name just a few [37][38][39][40][41]. In principle they can implement almost arbitrary sequential behaviour [42][43][44]. Furthermore, there has been interest, in recent years, in the observed ability of RNNs to model multiple nonlinear systems with fixed weights (topic covered in chapter 3).

3.2 Artificial Neural Networks

The main branch of Artificial Intelligence research in the 1960s -1980s produced Expert Systems. It became rapidly apparent that these systems, although very useful in some domains, failed to capture certain key aspects of human intelligence. In order to reproduce intelligence, it would be necessary to build systems with a similar architecture like a brain.

The basic units of the brain are the individual nerve cells "neurons". There are about 10^{11} neurons in the human brain massively interconnected (with an average of several thousand interconnects per neuron, although this varies enormously). Each neuron is a specialized cell, which can process and propagate an electrochemical signal. All neurons have the same basic morphology, and consist of a soma, or cell body, which is the central part of the cell between input and output branching structure: dendrites and the axon. The axon of one cell connects to the dendrites of another via a synapse. Through these synapses, neurons can communicate with each other. When a neuron is activated, it sends out spikes of electrical activity through the axon to other neurons. A neuron is activated only if the total signal received at the cell body from the dendrites exceeds a threshold.
An Artificial Neural Network (ANN) is an information-processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. ANNs can be described as computational structures built up from highly interconnected neurones (I will use the term units) working in parallel to solve a specific problem. However, much is still unknown about the brain (even at the lowest cell level) that the models used for ANNs seem to introduce an oversimplification of the 'biological' models.

3.2.1 Training of ANNs

Learning in biological systems occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. This is true in ANNs as well. Every artificial neural network possesses knowledge, which is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. One way is to set the weights explicitly, using a priori knowledge. Another way is to "train" the neural network by feeding it teaching patterns and letting it change its weights according to some learning rules. In this case, the teaching patterns must be selected carefully otherwise the network might be functioning incorrectly.

Learning methods can be classified into two major categories: distinction we can make is between:

Supervised learning in which the network is trained such that it more or less precisely reproduces training data provided by an external teacher, hoping that it then generalizes to new inputs.

Unsupervised learning also refered to as self-organisation in which a (ouput) unit is trained to respond to clusters of pattern within the input. Unlike the supervised learning, no target values are involved; rather the network self-organizes data presented and detects their emergent collective properties.

3.2.2 Neural Networks Topologies

The arrangement of neural processing units and their interconnections can have a profound impact on the processing capabilities of the neural network. The main distinction we can make is between:

Feedforward neural networks where the data flow from input to output units is strictly feedforward(figure (3.1.a)). The data processing can extend over multiple (layers of) units, but no feedback connections are present.

Recurrent Neural Networks (RNNs) which contain feedback connections. Contrary to feed-forward networks, the network activation produced by past inputs can cycle back and affect the processing of future

inputs. RNN topologies range from partially recurrent (figure (3.1. b) to fully recurrent networks (figure (3.1. c))

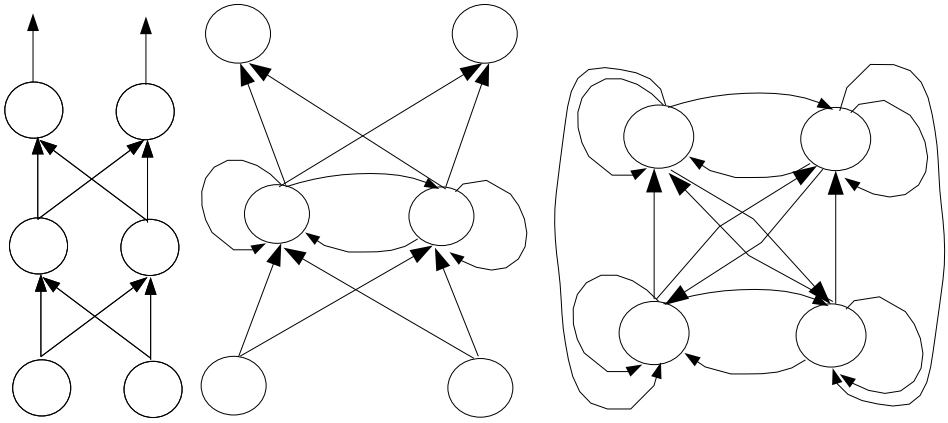


Figure 3.1: Artificial Neural Networks. a) Feedforward Neural Network. b) Partially connected RNN. c) Full connected RNN.

3.3 Learning in Recurrent Neural Networks

The dynamics presented by a RNN can be continuous or discrete in time. A continuous-time RNN is a complex nonlinear dynamical system described by a set of nonlinear differential equations. This can be generally expressed in the following form:

$$\dot{X}(t) = -\alpha X(t) + f(W, X(t), W_i, U(t)) \tag{3.1}$$

where $X = [x_1, x_2, \ldots, x_N]^t \in \Re^N$ and $U = [u_1, u_2, \ldots, u_K]^t \in \Re^K$ are the neural state and the input vector, respectively, $W \in \Re^{N \times N}$, $W_i \in \Re^{L \times K}$ are the connection weight matrices associated with the neural state and the input vector, respectively. The parameter α is a fixed constant and is chosen as $0 < \alpha < 1$, and $f : \Re^N \times \Re^K \rightarrow \Re^N$ is an appropriately chosen vector-valued nonlinear function.

Discrete-time models are described by difference equations:

$$X(k+1) = f(WX(k) + W_i U(k+1)) \tag{3.2}$$

where $X = [x_1(k), x_2(k), \ldots, x_N(k)]^t$ is the state vector, $U = [u_1, u_2, \ldots, u_K]^t$ is the input vector, $W = [w_{ij}]_{N \times N}$ is the connection weight matrices associated with the neural state and $W_i \in \Re^{L \times K}$ is the connection weight matrix with the input vector.
Training recurrent networks is often found to be problematic. The first difficulty is associated with the derivative calculations that are required for gradient-based training procedures. Due to the dynamical nature of these networks, the derivatives must reflect the network dynamics. Treating these derivatives as static functions of the network weights will result in poor mapping performances, since the weights search will be along inappropriate directions. Several methods have been explored for calculating derivatives with respect to network weights. Two approaches are generally used: Back-propagation Through Time (BPTT) and Real-time Recurrent Learning (RTRL).

3.3.1 Backpropagation Through Time

The central idea of BPTT, proposed by Werbos [45], is the unfolding of the discrete-time recurrent neural network into a multilayer feedforward neural network (FFNN) each time a sequence is processed. Instead of mapping a static input to a static output, BPTT maps a series of inputs to a series of outputs. This provides the ability to solve temporal problems by extracting how data changes over time.

To illustrate this idea in general, consider the RNN shown in Figure(3.2.a), which has n units fully connected and w_{ij} is the weight associated from the unit i to j.

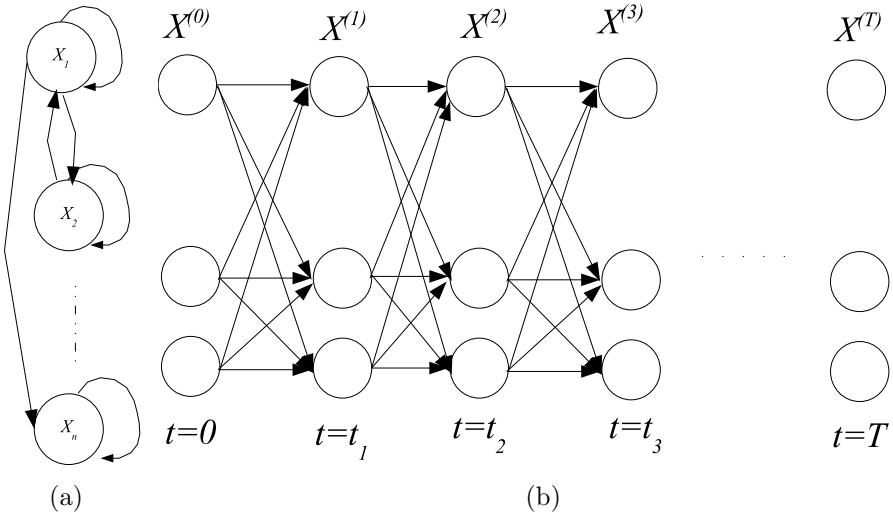


Figure 3.2: Backpropagation Through Time. a) recurrent neural network. b) network unfolded in time.

By unfolding the network of figure (3.2.a) at the time steps $1, 2, \ldots, T$, it can be regarded as one sweep of operation in a T-layered feed-forward network with identical connection weights w_{ij} between successive layers (Fig.3.2.b). This new representation is amenable to the backpropagation algorithm.

Let the training data set be partitioned to epochs. Let t_0 denote the start time of an epoch, and T its end time. Assume that desired responses specified for a set of units is D, and $e_j(t)$ is the error at the output of such a unit. The error to be minimized is:

$$E_{total}(t_0, T) = \frac{1}{2} \sum_{t=t_0}^{T} \sum_{D} e_j^2 \tag{3.3}$$

The BPTT algorithm is described as follows: Starting with an initial state $X(0)$, a single forward pass through the network for the interval $[t_0, T]$ is performed. At each time t, the complete record of input data, network state, and desired responses are stored. A single backward pass over this record is performed to compute the values of the local error gradients. These gradients can be derived in a same way as in the standard back-propagation, except that the output errors are not only given in the last layer but also added in each layer:

$$\delta_i(t) = -\frac{\partial E_{total}(t_0, T)}{\partial \nu_j(t)} \tag{3.4}$$

where $\nu_j(t)$ is the output of unit j, before the activation function. $j \in D$ and $t_0 < t \leq T$.

$\delta_i(t)$ is computed using the equations:

$$\delta_i(t) = \begin{cases} f'(\nu_j(t))e_j(t) & if \quad t = T \\ f'(\nu_j(t))[e_j(t) + \sum_{k \in D} w_{kj}\delta_k(t+1)] & if \quad t_0 < t < T \end{cases} \tag{3.5}$$

where $f'(.)$ is the derivative of an an activation function with respect to its argument.

Ones the computation above are repeated step by step from $t = T$ to time step $t = t0 + 1$, the connection weights adjustment is made according to

$$\Delta w_{ji} = -\eta \frac{\partial E_{total}(t_0, T)}{\partial w_{ji}} \tag{3.6}$$

$$\Delta w_{ji} = \eta \sum_{t=t_0+1}^{T} \delta_j(t) x_i(t-1) \tag{3.7}$$

where η is the learning-rate parameter and $x_i(t-1)$ is the i^{th} input of unit j at time $t-1$.

It is well known that the standard backpropagation convergence is slow. This remark is also true for the BPTT. The computation of one epoch is $O(TN^2)$, where N is the number of units. Thousands of epochs are typically required to a given problem. So, BPTT is not suitable for real time operation of a recurrent network.

3.3.2 Real-Time Recurrent Learning

Here, we describe another learning algorithm for a RNN that runs continuously. The network so trained is called a *real-time recurrent network*. Real-time recurrent learning (RTRL) has been independently derived by many authors, although the most commonly cited reference for it is Williams and Zipser [42].

Consider a network consisting of a total of N units with M external input

connections. Let $\mathrm{u}_{ext}(t)$ denote the M-by-1 external vector applied to the network at discret time t, and let $\mathrm{y}(t)$ denote the corresponding N-by-1 output units. Let W denote the N-by-$(M+N)$ recurrent weight matrix of the network. The matrix W consists of two sets of weights. A set, where the weights represent the connection between $\mathrm{u}_{ext}(t)$ and the network units, i.e. $w_{ij} \forall i \in U$ and $j \in I$, where U and I are the set of the network units and the external inputs, respectively. The other set is formed by weights that represent connections of the recurrent path, i.e. w'_{ij} $(\forall i, j \in U)$.

Let D denote the set units with teaching status, and $d_k(t)$ denote the desired response of the actual unit output $y_k(t)$ at time t, where k belongs to the set D. The main objective of RTRL algorithm is to minimize the network error $E(t)$ at time t through updating the weight matrix W as follows:

$$E(t) = \frac{1}{2} \sum_{k \in D} e_k^2(t) \tag{3.8}$$

where $e_k(t) = d_k(t) - y_k(t)$.

For each learning cycle, we may thus define the incremental change $\Delta w_{ij}(t)$ made at time t as follows:

$$\Delta w_{ij}(t) = -\eta \frac{\partial E(t)}{\partial w_{ij}} = \eta \sum_{k \in D} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}} \tag{3.9}$$

and

$$\Delta w'_{ij}(t) = -\eta \frac{\partial E(t)}{\partial w'_{ij}} = \eta \sum_{k \in D} e_k(t) \frac{\partial y_k(t)}{\partial w'_{ij}} \tag{3.10}$$

where η is the learning rate. Then, the weights are updates by

$$w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t) \quad \forall i \in U, j \in I. \tag{3.11}$$

and

$$w'_{ij}(t+1) = w'_{ij}(t) - \Delta w'_{ij}(t) \quad \forall i, j \in U. \tag{3.12}$$

We may now define a triply indexed set of variables $\pi_{ij}^k(t) = \frac{\partial y_k(t)}{\partial w_{ij}}$ and $\pi_{ij}^{\prime k}(t) = \frac{\partial y_k(t)}{\partial w'_{ij}}$, which represent the learning sensitivities of the units with respect to their weight connections. They are updated by

$$\pi_{ij}^k(t+1) = f'[S_k(t)][\sum_{l \in U} w_{kl}(t) \pi_{ij}^l(t) + \delta_{ik} u_{extj}(t)] \quad \forall i \in U, j \in I \ \ and \ \ k \in U. \tag{3.13}$$

and

$$\pi_{ij}^{'k}(t+1) = f'[S_k(t)][\sum_{l\in U} w'_{kl}(t)\pi_{ij}^{'l}(t) + \delta_{ik}y_j(t)] \quad \forall i,j,k \in U. \tag{3.14}$$

where

$$S_k(t) = \sum_{l\in U} w_{kl}u_{extl}(t) + \sum_{l\in I} w'_{kl}y_l(t) \tag{3.15}$$

δ_{ik} is the Kronecker delta, and $f'(.)$ denotes the derivative of the sigmoid function $f(.)$. RTRL is mathematically suitable for online training. However, its computation cost is expensive, regarding to the size of the network. In the case of an n units fully network, the complexity computation is $O(n^4)$. Thus this algorithm is useful for on-line adaptation only if a small network is sufficient to solve a given problem. Schmidhuber [44] proposes a method, which combines BPTT and RTRL to reduce the average time complexity per time from $O(n^4)$ to $O(n^3)$.

3.3.3 Difficulty of learning long-term dependencies.

BPTT, RTRL and their combination share an important difficulty. It is found in practice, that the network activity from time long past has less effect than the current network activity for the current derivative calculations. It is shown in [46] that the temporal evolution of the back-propagated error depends on the magnitudes of the weights involved. Depending on the gain of the network loop, the gradient will either explode exponentially or vanish [47]. Since the networks are usually initialized with small weights using sigmoidal activation function with small derivatives, most of the gradients decay in time. Hence standard RNNs fail to learn in the presence of long time lags between early inputs and late desired outputs. Several solutions have been proposed to overcome this difficulty. One is the use of long short-term memory (LSTM), proposed by Hochreiter and Schmidhuber [47].

3.3.4 Long Short-Term Memory

The objective of LSTM is to retain important information over a much longer period of time than the 10 to 12 time steps, which is the limit of RTRL or BPTT models. The basic idea is to replace a hidden layer of traditional RNN by a memory bloc. Figure (3.3) shows a simple LSTM network, with a single input, a single output, and a single memory block in place of the

familiar hidden unit. The memory bloc contains linear units with fixed self-connections, called memory cells. They solve the vanishing error problem by enforcing error signals in the cell to remain constant, neither growing nor decaying. Multiplicative adaptive gate units control access to the memory cells. They are conventional units with sigmoidal activation functions ranging over $[0, 1]$, and they receive input from the network input units and from other cells. If the gate activation is near zero, nothing can enter the cell. Similarly, nothing emerges from the cell unless the output gate is active. Input and output gates protect the memory cells from irrelevant inputs and noise, and permit to do not perturb the remainder network. Network training uses a combination of RTRL and BPTT training algorithms.

LSTM algorithm has been applied successfully to wide range of tasks; i.e. the embedded reber grammar benchmark [47], speech recognition [48], and music improvisation [49].

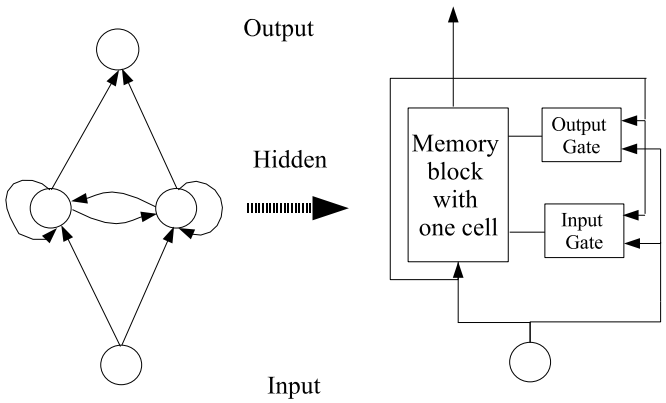


Figure 3.3: Left: A fully recurrent hidden network. Right: LSTM network with a memory bloc.

3.3.5 Extended Kalman Filter for RNNs Weight Estimation

The gradient descent is not only slow, but also ineffective to find suitable solutions. This difficulty is addressed by Feldkamp et Puskorius [50] [51] via the Extended Kalman Filter (EKF). It is found that EKF-based training procedures proved superior performances than simple gradient methods for many categories of problems.

The EKF is an estimation technique for nonlinear dynamics and nonlinear measurement equations. It is an optimal estimator that recursively combines noisy sensor data with a model of the system dynamics. Training neural networks using kalman filter was first proposed by Singhal and Wu [52]. They considered the network weights as a state of a dynamical system to be estimated, assuming that the transient effects of the network state have died out. Using the EKF algorithm, the weights updates become [53]:

$$K(t) = P(t)H(t)[\frac{1}{\eta}I + H(t)^t P(t)H(t)]^{-1} \tag{3.16}$$

$$W(t+1) = W(t) + K(t)e(t) \tag{3.17}$$

$$P(t+1) = P(t) - K(t)H(t)^t P(t) + Q(t) \tag{3.18}$$

Where $e(t)$ is the difference between the observed and the desired network output. The matrices H contain the derivatives of the network outputs with respect to the weights, and $P(t)$ is an estimate of the error covariance. $K(t)$ is the Kalman gain matrix used in updating the network weights W. The learning rate η in (3.16) is essential to compensate possible bad initialisation of $P(t)$. Each approximate error covariance is augmented with a diagonal matrix $Q(t)$ that represents effects of the process noise. This noise insertion improves the algorithm's numerical stability and avoids poor local minima [53].

3.3.6 Temporal Integration in RNNs

Biological neural networks are obviously recurrent, and the existence of feedback connections on several spatial scales is one property of the brain. Maass et al.[54] propose that there is a link between adaptive real-time response in different cortical areas of the brain and temporally integrated information. Adaptive real-time response means the ability to produce at any time t a meaningful response that depends on inputs that the network received at various times $s < t$ back in the past. Such "any-time intelligent" and meaningful response is desirable for artificial neural networks. The most known approach to make past inputs $u(s)$ available for the current output response $y(t)$ is to maintain past samples the inputs $u(t-\Delta), u(t-2\Delta), \ldots, u(t-k\Delta)$ that arrived at a fixed number k of discrete time points. This memory window has to be updated after every time interval Δ. An obvious disadvantage of this strategy is the rigid prescribed length of past samples of u, and the

rigid prescribed sampling period Δ. Since a number of input delays may be adequate for some computational tasks, but not for others. Furthermore, there is no evidence that this strategy can be used for temporal integration tasks that require temporal integration over a few hundred milliseconds (ms) and longer [54].
Another strategy is to condense all information from earlier inputs into the current internal state $x(t)$ of the network. This condensed information might be needed for a decision at time t. An advantage of this strategy is that the temporal integration is not restricted to a fixed sampling interval Δ for past inputs, nor to a fixed number of past samples. The problematic in this approach is to find a predefined set $\aleph$ (or attractors) of states $\mathrm{x}(t)$. Furthermore, such attractor neural networks are in general not able to produce at any time t a meaningful response $y(t)$ from past input samples because they spend most of their time in transient states between attractors. Moreover, study of dynamical systems formed by attractor neural networks focused on autonomous dynamical systems (i.e., systems whose input can be encoded in the initial state of the system) and usually consider a very low dimension [54].

An alternative method [55] to attractor neural networks is to extract information during transient dynamics of a recurrent network, without controlling or manipulating the transient dynamics. This method does not require convergence to stable internal states, since information about past inputs is captured during the continuous trajectory of transient internal states [56]. It only remains to read out this information. It is shown in [55] that a readout unit can learn to extract salient information from a high dimensional transient state of a large RNN, and can transform this transient state into a stable readout. This readout unit can be trained by supervised or unsupervised learning methods to perform a specified task. This model that analyze computations without stable states is called *Liquid State Machine* (LSM). A Similar idea has been discovered independently by Herbert Jaeger [57], called Echo State Network (ESN). We will present first the principals of LSM, then we will give more detailed information about ESN.

3.3.7 Liquid State Machine

Liquid State Machine (LSM) was discovered by Wolfgang Maass et al. [55]. The idea is inspired from the behaviour of a liquid exited by external perturbations (inputs) such as wind, sound ... etc. Maass et al. propose that "...

the perturbed state of the liquid, at any moment in time, represents present as well as past inputs, potentially providing the information needed for an analysis of various dynamic aspects of the environment ...". Their approach is based on the following observations. If a continuous input stream $u(s)$ excites a large "pool" of randomly connected units, then after a later time $t > s$ the current state of the network, also called "liquid state", $x(t)$ holds a substantial amount of information about recent inputs $u(s)$ [58]. The objective is then to extract this information. LSM can be considered as a filter that maps the input $u(.)$ onto an output $y(.)$. Figure(3.4) illustrates the basic architecture of the LSM.

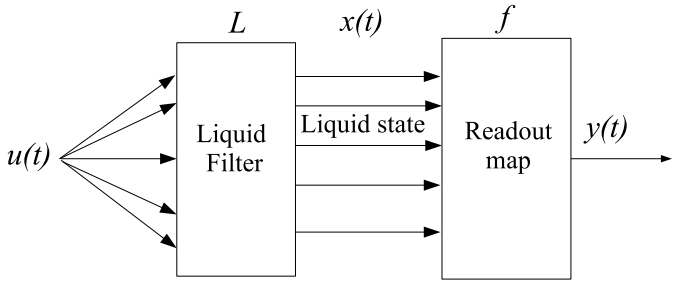


Figure 3.4: Basic architecture of Liquid State Machine.

The first component of LSM is an operator L that maps input functions $u(.)$ onto liquid state $x(t)$:

$$x(t) = (Lu)(t) \tag{3.19}$$

where L is referred to "liquid filter".
The second component is a memory-less readout map f that transforms the current liquid state $x(t)$ into the output $y(t)$ according to:

$$y(t) = f(x(t)) \tag{3.20}$$

The readout map f is adjusted to the specific task. It is chosen as memory-less, because as cited above, the current liquid state $x(t)$ contains all information about inputs $u(s)$ from preceding time $s \leq t$ that is needed to produce the output $y(t)$.

An implementation of LSM was carried out using a randomly connected recurrent network called "liquid neurons" as a liquid filter. Liquid neurons are based on integrate-and-fire neurons. The output at time t of all the liquid neurons represented the liquid state of a LSM. The readout map f was

formed by other unconnected integrate-and-fire neurons with a biologically realistic membrane time constant of $30ms$ that are trained to approximate the target value of the output $y(t)$. Simulations reported in [55] demonstrate the computational universality of generic recurrent networks of integrate-and-fire neurons, if viewed as special cases of LSMs.

3.4 Echo State Network

Echo State Network (ESN), developed recently by Herbert Jaeger [57], has the same philosophy as a liquid state machine: i) Both, ESN and LSM, are three-layered networks. ii) In both approaches, the hidden layer uses a recurrent network with a stochastic connectivity. iii) They have the same strategy of learning, where only the readout map is to be adjusted according to the task at hand. However, ESN and LSM are different in the nature and in the objective. The "liquid state" in LSM is made by continuous model with biologically inspired integrate-and-fire units, whereas the so-called "reservoir" of ESN is made by large number of sparsely interconnected simple sigmoid units. As seen above, LSM research focuses on modelling and exploring biological neural networks dynamics, whereas ESNs are oriented to engineering applications (prediction, modelling, adaptive control ...etc).

3.4.1 Formal Description

Let us now turn to a more formal description of ESNs. The notation will be similar to the notation used in [57]. As presented in Figure(3.5), Echo state network is formed by a so-called "Dynamic Reservoir"(DR), which contains a large number of sparsely interconnected units with non-trainable weights. We consider that the network has K inputs, N internal units and L output units. Activations of input units at time step n are $U(n) = (u_1(n), u_2(n), \ldots, u_k(n))$, of internal units are $X(n) = (x_1(n), \ldots, x_N(n))$, and of output units are $Y(n) = (y_1(n), \ldots, y_L(n))$. Weights for the input connection in a $(N\text{x}K)$ matrix are $W^{in} = (w_{ij}^{in})$, for the internal connection in a $(N\text{x}N)$ matrix are $W = (w_{ij})$, and for the connection to the output units in an L x $(K+N+L)$ matrix are $W^{out} = (w_{ij}^{out})$, and in a $(N\text{x}L)$ matrix $W^{back} = (w_{ij}^{back})$ for the connection from the output to the internal units.

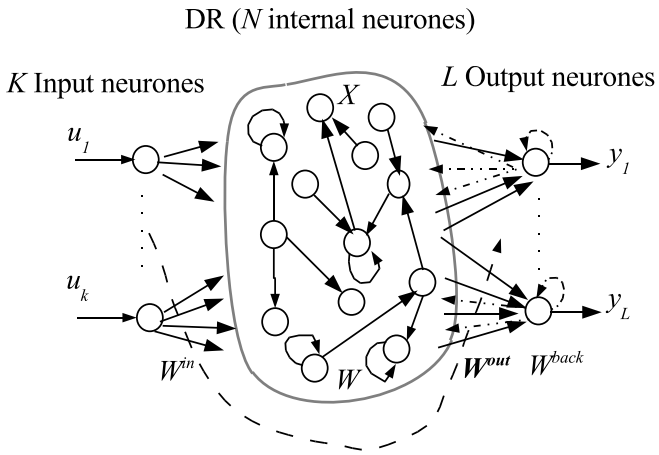


Figure 3.5: Basic architecture of ESN. Dotted arrows indicate connections that are possible but not required.

The activation of internal and output units is updated according to:

$$X(n+1) = f(W^{in}U(n+1) + WX(n) + W^{back}Y(n+1)) \tag{3.21}$$

where $f = (f_1, \ldots, f_N)$ are the internal units output sigmoid functions.

The outputs are computed according to:

$$Y(n+1) = f^{out}(W^{out}(U(n+1), X(n+1), Y(n))) \tag{3.22}$$

where $f^{out} = (f_1^{out}, \ldots, f_L^{out})$ are the output neurons output sigmoid functions. The term $(U(n+1), X(n+1), Y(n))$ is the concatenation of the input, internal, and previous output activation vectors.

The idea of this network is that only the weights connections from the internal units to the output (W^{out}) are to be adjusted.

3.4.2 How ESN approach works?

The key to understanding ESN approach is the concept of *echo states*. It is a property of the network prior to training. Echo states property is relative to the weight matrices (W^{in}, W, W^{back}) and to training data.

Compactness conditions

Compactness conditions are resumed as following: i) If input-ouput sequences are drawn from spaces U and D respectively, then it is required that U and D be compact. ii) During update, network states are required to stay in one compact set ($A \subset \Re^N$). It means that if at time n, the initial state $X(n) \in A$, then after T iteration updates of (3.21) $X(n+T) \in A$.

Definition 1 *(echo states) Assume that the compactness conditions are verified. Then a network* (W^{in}, W, W^{back}) *has echo states with respect to the compact intervals* U *and* D*, if the network state* $X(n)$ *is uniquely determined by the history of the input-output data.*

An equivalent way of stating echo state property is to say that for each internal state x_i there exists a function $e_i : (U \times D)^{-N} \rightarrow \Re$ that maps the input-output history to the current state:

$$x_i(n) = e_i(\ldots, (u(n-1), d(n-2)), (u(n), d(n-1))). \tag{3.23}$$

Several equivalent characterizations of echo states are collected in [59].

Unfortunately, there is no known necessary and sufficient condition that permits to say whether a network has the echo state property. There is, however, a sufficient condition for the non-existence of echo states. I quote here this condition from [57]:

Proposition 1 *Assume an untrained network* (W^{in}, W, W^{back}) *with state update according to equation (3.21) and with transfer functions tanh. Let* W *have a spectral radius* $|\lambda_{max}| > 1$*, where* $|\lambda_{max}|$ *is the largest absolute value of an eigenvector of* W*. Then the network has no echo states with respect to any input/output interval* $U \times D$ *containing the zero input/output* $(0, 0)$*.*

This proposition is not helpful for finding "constructing" echo state networks. However, in practice it is found that when the spectral radius $|\lambda_{max}| < 1$, we do have an echo state network. This observation and proposition 1 still need mathematical analysis.

Definition 2 *A network* $(W^{in}, W, W^{out}, W^{back})$ *is an echo state network if its untrained "core"* (W^{in}, W, W^{back}) *has the echo state property with respect to the compact intervals* U *and* D*.*

3.4.3 Training Algorithm

Here we will give a detailed description of training an ESN $(W^{in}, W, W^{out}, W^{back})$ for a given task. The goal is to adjust (train) the matrix W^{out} such that the network output $y(n)$ approximates a desired output $d(n)$, when the network is driven by training input $u(n)$.

Step 1: echo state property

The following procedure seems to give a practical solution to guaranty echo state property:

1. The order of input and output units should be fixed according to the task at hand.
2. Generate randomly the input weights W^{in} and output backpropagated weights W^{back}.
3. Generate randomly an internal weight matrix W_0.
4. Normalize W_0 with its spectral radius λ_{max} and put it in $W_1 : W1 = \frac{1}{|\lambda_{max}|} W_0$.
5. Scale W_1 with a factor α and put it in the final internal matrix of the network $W : W = \alpha W$, where $0 < \alpha < 1$.

It has been always found that using the steps above the untrained network (W^{in}, W, W^{back}) is an echo state network.

Step 2: driving the network by training data

Ones the echo state property is verified, the ESN should be driven by given I/O training sequence $(u(n), d(n))$. It is a mechanical step, which involves the following operations:

1. Compute the network states by presenting I/O training data:

$$X(n+1) = f(W^{in}U(n+1) + WX(n) + W^{back}Y(n+1)) \qquad (3.24)$$

$n = 0, \ldots, T$

2. Collect at each time the state $X(n)$ as a new row into a state collecting matrix M, and collect similarly at each time the sigmoid-inverted teacher output $tanh^{-1}D(n)$ into a teacher collection matrix C. After these collections, the matrix M has the size of $(T+1) \times (K+N+L)$, and the matrix C has the size of $(T+1) \times L$.

Remark 1 *Sometimes it is desirable to don't consider the transient phase of the network during training. This is easily made by washing out data, which are collected before a certain time T_0. Thus, the sizes of M and C will be $(T-T_0+1) \times (K+N+L)$ and $(T-T_0+1) \times L$, respectively.*

Step 3: output weights computation

Compute the pseudoinverse of M and put:

$$W^{out} = (M^{-1}C)^t \tag{3.25}$$

t: indicates transpose operation.
Computing pseudoinverses exists in every programing package of numerical linear algebra.

Step 4: exploitation

The ESN is now trained. For exploitation, the network can be driven by new input sequences and using the equations (3.21) and (3.22) repeated here for convenience:

$$X(n+1) = f(W^{in}U(n+1) + WX(n) + W^{back}Y(n+1)) \tag{3.26}$$

$$Y(n+1) = f^{out}(W^{out}(U(n+1), X(n+1), Y(n))) \tag{3.27}$$

After several implementations of ESNs on real systems, I want to give some benefit remarks:

1. The matrix W^{in} plays an important role on the behavior of the ESN. Small absolute values of W^{in} mean that the network internal units operate around the linear part of the sigmoid, i.e it is almost a linear dynamics. With large absolute values, the network will be strongly

driven by the input, and therefore the internal units will operate closer to the saturation of their sigmoid, which result to a more nonlinear behavior of the network. Similar remarks hold for W^{back}.

2. Keep the connectivity sparse inside the internal matrix W_0, like in biological neural systems, in order to avoid chaotic effects. In my experiments, 10% and 20% of connectivity works well.

3. There is no general rule to find an optimal size of W_0. In my experimental implementation on DC motors and mobile robots, an ESN controller, with more than 30 internal units, lost stability at many times. This is due perhaps of the high degree of freedom of the closed loop. The optimal method is to begin with a small network (say 9 units), and after each test increase the number of internal units until an acceptable behavior is reached.

4. The parameter α needs to be also hand-tuned. It should be small for fast dynamics and large for slow dynamics.

3.5 Conclusion

This chapter discussed a variety of theoretical and practical RNN learning approaches. The intensity of recent work in this field shows that it is worth to deal with their computational complexity in order to profit from their powerful capabilities. Some well settled issues can be identified: gradient learning-based are well understood but they suffer from some problems: i) local minima, ii) slow convergence, iii) difficulty of Learning Long-term dependencies. Furthermore, they are relatively complex to implement. These reasons might explain why RNNs are used less often for real-world applications than FFNNs, although these methods have been known as long as the backpropagation algorithm for FFNNs. To overcome these difficulties, many researchers propose to use either task specific algorithms, or to use specialized architectures, like Long short term memory networks, or both.

Recently, random recurrent neural networks were proposed to avoid the difficulties involved with training recurrent neural networks. The idea is that a large recurrent network can serve as a "source of dynamics" from which information can be extracted by a readout unit. One approach "liquid state machine (LSM)" is biologically motivated and uses continuous time spiking

units [55]. In that work, results show that under some assumptions the network considered is computationally universal and demonstrate some biologically motivated tasks. A similar approach "echo state network (ESN)" was discovered independently by Jaeger [57]. Many dynamical systems, which were difficult to learn with the existing methods, have been easily learnt by ESN [60][59].
In this thesis, ESN performances will be used in many non-trivial tasks: Meta-learning, adaptive nonlinear system identification, high- and low-level adaptive control of mobile robots, and real implementations on an omnidirectional mobile robot.

Chapter 4

Metalearning

Meta-learning research deals with the question: *how learning algorithms can improve their performances through past knowledge*? The machine learning community uses meta-learning to build up neural networks, which have the ability to adapt without changing explicit weights. This ability is acquired by construction (learning learning rules) or through prior training. One may directly ask: *How does the network adapt, if its weights are fixed*? It seems that this question was first discussed in 1990 by Cotter and Conwell [61]. In that work, they prove the Fixed Weight Learning Theorem that describes how fixed-weight of a RNN can approximate the dynamics of a feedforward network trained by an adaptive weight-learning algorithm. This approximation doesn't require any weight change, hence the "fixed-weight" label. Later on, several researchers have explored independently this notion, and referred it as *metalearning* or *learning how to learn*. Metalearning may be the most ambitious goal in artificial neural networks. Build a network, which has the ability to adapt without changing explicit weights, is very promising for non-stationary time series, robust identification and control of time varying systems, autonomous intelligent robots...etc.

This chapter consists of four sections. The next two sections introduce the notion of meta-learning and review recent results on adaptive behaviour attained with fixed weights recurrent neural networks. Section (4.3) describes

adaptive identification of non-linear dynamical systems problems and their solution with a fixed-weight echo state network. Finally, section (4.4) concludes with comments.

4.1 Review of Metalearning

It seems that metalearning represents a big puzzle. Each research group see metalearning from a different angle (for a comprehensive survey refer to [62]). Despite these differences, a question remains constant: *how can we improve learning algorithms by exploiting past knowledge (learning experience)*? In spite of the many research efforts, no clear answer has emerged. This section provides different views and definitions of metalearning as reported in the machine-learning literature.

First, it is important to give some basic concepts that will be helpful to present recent development in metalearning. Assume that we have to train a learning algorithm[1] L on a set of training examples $T_{train} : \{(Ui, Yi)\}_{i=1}^{n}$ where each object U_i is labeled with a class Y_i according to an unknown function F, $F(U_i) = Y_i$. Figure (4.1.a) shows a base-learning strategy usually used in a common machine learning procedure. A set of observed data from the task T is used to train the learner L, which, after being learned, will predict something about this task. As pointed in recent machine learning research, learning is a search process caried out by a learner over a space of hypothesis H, in order to derive a hypothesis h, $h \in H$ that could be useful in estimating a given concept [62][64]. To perform search in our example, the learning algorithm L maps the space of all training sets (Γ) into the hypothesis $L : \Gamma \rightarrow H$. The selected hypothesis h is used to approximate the function F, and then to make generalization on unseen examples. However, the common learning procedures proposed in machine learning (base-learning) are characterized by fixed bias[2]. A hypothesis space H depending on such learner will be also fixed, thus the learner is specialized in a limited fixed region R_L in S , where S is the space of all possible tasks (target function F, different training data size, different sampling period of training

[1] Algorithms that modify theirs modifiable components (policy) are called learning algorithms [63].

[2] All asymptions related to the behavior of a learner is refered to the "bias" term. This term can be considered as a tool of generalization and can be used to measure the power of a particular learning algorithm [65].

data,...etc)(Figure 4.1.b). The learning algorithm L is suitable only for those tasks inside R_L, and can never learn all possible tasks (i.e. T_e) in S as long as its bias is fixed [62]. It is interesting if through its past learning experience, the learning algorithm L changes its learning mechanism according to the task under analysis; this is what we call *learning to learn*. Following the terminology in [66], I will refer to the problem of dynamic selection of bias as the *meta-level* learning problem. The conventional learning problem will be referred to as the *base-level* learning problem (Figure 4.1).

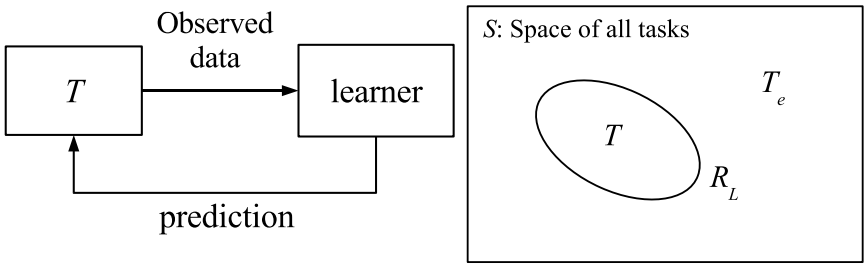


Figure 4.1: Base-level learning

4.1.1 Inductive Transfert

When learning tasks are closely related, it seems reasonable to expect a learner to perform increasingly better on a particular task by reapplying knowledge gaind in previous learning tasks. To improve learning through time, the learner must be able to transfer knowledge about learning (meta-knowledge) across tasks. This process is known as inductive transfer [67][68]. Inductive transfer is widely used in the realm of neural networks.

4.1.2 Dynamic Selection of Bias

Transferring knowledge across learning tasks involves dynamic selection of Bias. In Figure (4.1.b), the task T_e is situated outside the area of expertise, thus a base-learner L is not able to learn this task as long as its bias is fixed. This is common in machine learning, where a learner performs well in some contexts, but inadequat in others. To overcome this limitation, the base-learner should adapt its internal mechanism according to the task. This enable the learner to shift its region of expertise along the set of the

tasks. A metalearning approach would possibly able to extend the traditional base-learning mechanism from a fixed bias mechanism to a dynamical bias, providing a base-learner the ability to learn how to learn. One approach to solve this problem is to combine predictions from different learners, in order to shift the dominant region of the meta-learner over the task T_e [69][69]. A framework on dynamic-bias selection is given by DesJardins et.al. [65]. The authors propose three search tiers for a bias shifting system. The first tier refers to a search through a space H where a learner L derives the best hypothesis h, $h \in H$ that could approximate the desired task. Usually, most learning algorithms assume that this space is fixed. To perform a dynamic bias selection, the learner L must search in a second tier (bias search space), where the size of the hypothesis space H can be modified. In the third tier, search takes place at a meta-bias space level. At this level, it is possible to select a bias for any search space in the second tier.

4.1.3 Meta-learner of Base-learners

One objective in metalearning is to determine the properties of the base-learner L (interaction between its components) and how the properties of the tasks in R_L make L suitable in this region. Understanding the problem above permit to choose the right algorithm for a particular task.

Assume now that the task T lies inside intersection of k regions R_{L1}, R_{L2},..., R_{Lk}(Fig.4.2.a), where k learning algorithms L_1,L_2,... L_k are spesialized, respectively. In this case, it is reasonable to suppose that all learners are able to fit the task T. As mentioned above, one objective of metalearning is to use a meta-learner to combine predictions of base-learners in order to determine the best one suited for the task at hand (Fig.4.2.b). One common approach is the *Stacked Generalization* originally proposed by Wolpert [70]. In the example above, the k base-learners are applied to the training set $T_{train} : \{(Ui, Yi)\}_{i=1}^{n}$ to produce k hypotheses h_1, h_2,..., h_k, called level-0 generalizers. For metalearning, training data are transformed to new set T'_{train} , where data are replaced by predictions made by each hypothesis. The new training set T'_{train} serves as input to the meta-learner, which produces a new hypothesis called level-1 generalizer. The role of the meta-learner is then to select dynamicaly the best suitable learning algorithm for the task under study.

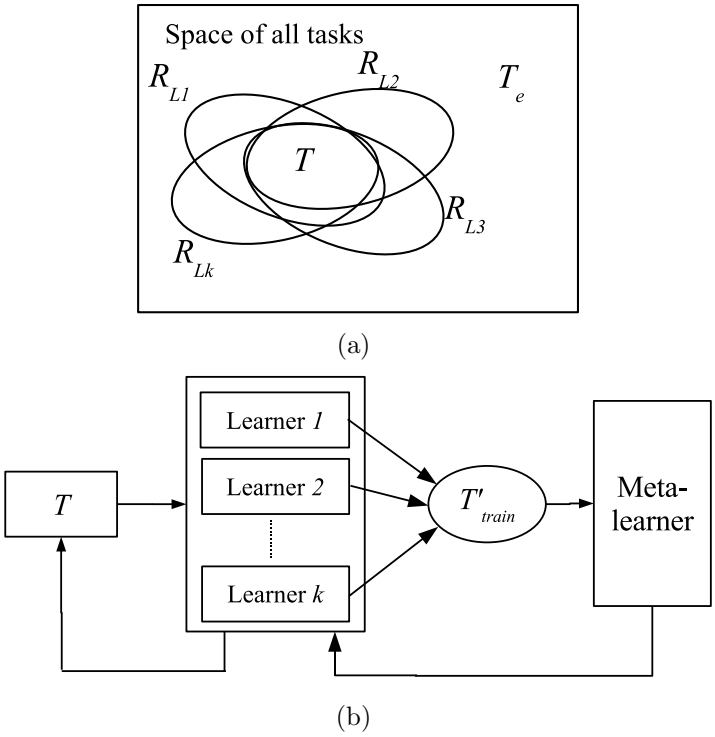


Figure 4.2: Meta-learning

4.1.4 Lifelong Learning

Common learning artificial neural network generalizes well if sufficiently many training examples are available. However, they often work poorly when training data are scarce. In contrast, humans often generalize accurately even in presence of scarce training data or from only a single training example. This is because of the re use of knowledge acquired in our previous lifetime.

Lifelong learning is a framework that addresses situations in which a learner faces many different learning tasks [71]. When facing a new learning task, the learner may transfer knowledge gaind in previous learning tasks to improve generalization. In the context of neural network, lifelong learning goes beyond the limited bounds associated with learning single functions in isolation. In [66], lifelong learning is investigated empirically in the context of object recognition. It is shown that in presence of scarce training data, some learning approaches that learn at the meta-level generalize significantly better than those that do not. In mobile robot navigation, Complex tasks require

huge amounts of training data when treated in isolation. These tasks can be achieved much faster if a robot can exploit previously learned knowledge [72].

4.1.5 Multitask Learning

Another way for the transfer of knowledge is concerned with the construction of better internal representations, which improve generalizations across multiples tasks. Caruana [73] shows that training multiple tasks on one learner improves generalization performances, since each task can benefit from the information contained in the training signals of other tasks. In that work, the *Multitask Learning* approach is used in FFNNs trained by backpropagation algorithm, where all tasks are treated equally. Many outputs (one for each task) share a common hidden layer (Fig. 4.3), and backpropagation is done in parallel on all outputs using the same inputs. Information stored in the hidden layer for one task can be used by other tasks.

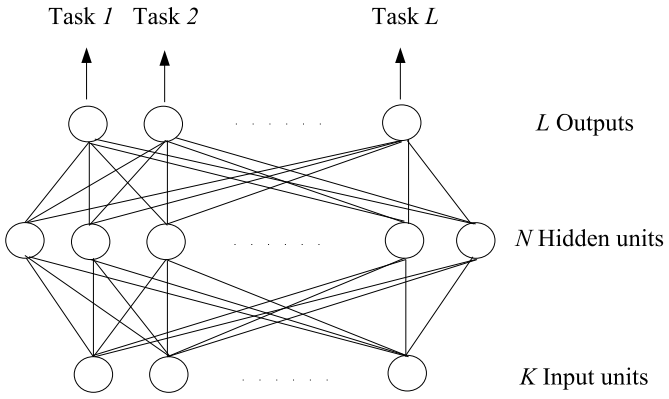


Figure 4.3: Multitask Learning of L tasks with the same inputs.

4.2 Fixed-Weight Neural Networks

The machine learning community uses metalearning to build up neural networks, which have the ability to adapt without changing explicit weights. This ability is acquired by construction (learning learning rules) or through prior training. While the first strategy focuses on the concept of learning to implement a learning rule, the second, instead, is built on the concept of

training a RNN that would approximate different families of functions. After an appropriate training of different functions, the obtained fixed-weight neural networks (FWNN) will be able to switch between them, using only input-output pairs from one function family, and without changing any synaptic weights. The two strategies principles are presented in this section.

4.2.1 Learning *Learning Rules*

Learning learning rules can be naively stated as following: *If neural networks can approximate functions, they can approximate learning rules.* The question is whether the learning rule can be considered as a well-behaved function. If so, then it is possible to built a network that learns this learning rule. To illustrate this idea we consider a one-neuron network (Figure (4.4)), which is asked to adapt with a fixed weight. Equation (4.1) gives its output y. Assuming that the weight is trained by a gradient descent, then the change in the weight w is given by the weight update rule (4.2), where u is the input, η is the step size, and $e = y - d$ is the error between the actual and the desired output. The weight is updated by (4.3).

$$y = tanh(wu) \tag{4.1}$$

$$\Delta w = -\eta E(e, u, y) \tag{4.2}$$

$$w_{k+1} = w_k + \Delta w \tag{4.3}$$

Inspired from the fixed-weight architecture proposed by Younger et al. [74], the one-neuron network will be transformed in a larger network (Figure 4.5). I will consider an auxiliary network called *weight update network* that learns (4.2). Equation (4.3) is realized by a single neuron that has a positive unit feedback connection. Finally, a Π unit replaces the original weight. The network of figure (4.4) is now transformed to a larger network, which after learning a *learning rule* can behave adaptively with fixed weight.

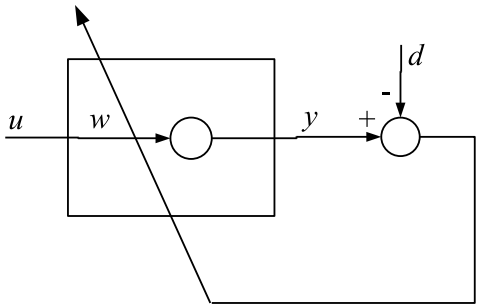


Figure 4.4: A weight adaptation of a single unit network.

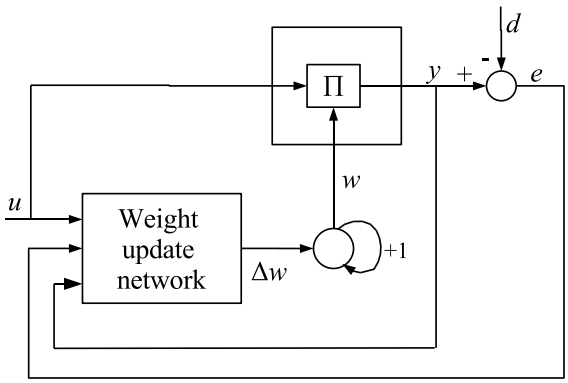


Figure 4.5: A fixed-weight network equivalent to a single unit network.

Schmidhuber in [63] proposes a method called self-modifying policy (SMP) to create learning learning algorithms. A reinforcement learning method called "success-story algorithm" (SSA) is used to force a SMP to trigger better and better self-modifications. In that work, a concrete implementation is presented, where SMP/SSA-based learners solve a complex task in a partially observable environment. Castiello et al. [64] propose a neuro-fuzzy system that improves its performance through past experience. The system integrates both a base-learner (performing the ordinary predictive tasks) and a meta-learner that supports the base-learner to search for a suitable bias in

every new specific task. In [75], a neural network is built to run and improve its own weight algorithm. Besides its learning capability to solve problems, the network uses its own weights as inputs and learns new algorithms for modifying its weights in response to the environment and evaluations of past learning performances.

4.2.2 Multiple Modeling

Other researchers focus on the concept of training fixed-weight RNNs that would approximate different families of functions. It is stated in [76] that any recurrent neural network is a potential metalearning system. In a RNN, recurrent signal loops can store information by accumulation, and they act like weights in a conventional network [74]. Therefore, adaptation in RNNs can manifest only by signal changing in recurrent loops, without need to change weights to react to a changing environment. The adaptive behavior with fixed weights is named differently. It is termed metalearning in [51], and "accommodative" in [77]. In this concept, the selection of training data determines the difference between base-learning and meta-learning. In a baselearning, training data are collected from a single functional mapping. For metalearning, training data are collected from different families of functional mappings such that, after learning, the FWNN produces the correct output as specified by the selected mapping. It has been shown by Feldkamp et al.[78] that a single fixed weight RNN can perform one-time-step prediction for many distinct time series. In the control domain, it is shown in [79] that a RNN can be trained to act as a stabilizing controller for three unrelated systems and to handle switch between them.
In some way, this adaptation depends on storing some meta-level information about the different functional mappings. This "meta-information" is explored as a form of memory in [80]. In that work, it is considered that a fixed-weight RNN encodes a set of memories. Each memory represents a certain mapping, which is already learned by the network. Adaptation is then defined as the ability of the fixed-weight RNN to recall from memory the information needed to adapt to the mapping function being presented to the network. Understanding how this recall mechanism occurs will help significantly to build more "intelligent" machine learning. Roberto et al. in [80] propose a reformulation of FWNN using a Context Discerning Multifunction Network (CDMN). CDMN is formed by a component called discernment network (CDN) that is able to identify spatial and temporal characteristics and

produce fixed parameters. These parameters can be used by another component called multifunction network (MFN) to adjust biases over the presented task (mapping). CDN and MFN form together the concept of CDMN.

A major problem to be adressed in metalearning is the *recency effect*: the tendency to forget what hat learned in the past. If training data are homogenous, then one or more passes through the data can probably produce good results. However, when training data are heterogeneous, the tendency of the weights update will be in a favour of the most recently presented data. The same phenomenon can be found in case of training feedforward networks if training data are repeatedly presented in the same order. To avoid this, an effective solution is to scramble the order of training data presentation, or to run the network through the entire data set, and make an update based on the entire set errors. The later is known as batch update algorithm.
Multi-stream procedure [81][53][80] is an extension of EKF training methods, which combines both scrambling and batch update methods. As a result, this procedure provides the benefits of the batch learning to circumvent the recency effect, while remaining consistent with the kalman recursion.

Multistream Training

We describe now the general idea of multi-stream training. In a problem of heterogeneous training data, usually we deal with more than one data file. Each file contains data collected from a different system, or from one system under different operating conditions. Multi-stream training is based on the principal that each weight update should satisfy simultaneously the conflict demands of heterogeneous training data. First, a specified number of randomly selected starting points is chosen in each file. Each starting point is the beginning of a stream. For each stream, a separate RTRL or a BPTT(h) derivative calculation is used, in order to carry out weight updating. After saving copies of output units and dynamics derivatives from each stream, a global EKF update procedure is used. If first-order updates were being used, one can simply average these updates. A more rigorous method is to consider the problem as that of training a multiple output network in which the original output unit's number is multiplied by the number of streams.

4.3 Adaptive Identification with Fixed-Weight ESN

In this section, we want to explore the ability of ESNs to exhibit adaptive behavior with fixed weights in system identification. This is not to demonstrate that the fixed-weights ESN can pass any test for adaptive systems. Rather, we wish to report the metalearning technique so that the resulting fixed-weight ESN identifier exhibits an adaptive behavior [82]. No multistreaming is needed, since the ESN has a batch learning algorithm and does not suffer from the recency effect.

4.3.1 Preliminaries on Nonlinear System Identification

System identification is extremely relevant in real implementations and only recently has much ongoing research addressed the problem of identifying systems with nonlinearities. In practice, these systems are frequently modelled as input/output (I/O) maps of sampled data from the system. A standard model that has been used to represent a wide class of nonlinear dynamic systems is the Nonlinear Autoregressive Moving Average (NARMA) model given by

$$d(k) = G(d(k-1), d(k-2), \ldots, d(k-n_d), u(k), \\ u(k-1), \ldots, u(k-n_u), \theta_k) \tag{4.4}$$

where $G(.)$ is some nonlinear function, θ_k denotes parameters with values from compact set Θ, and $(u(k), d(k))$ the input/output pair measured from the system at time k. The parameters n_d and n_u are the output and input orders.
The problem of nonlinear system identification is to approximate the function $G(.)$ in (4.4) from some sampled data sequences $\{(u(k), d(k))|k = 1, 2, \ldots, t\}$.

It is well known that system identification can be done using either parallel models or series-parallel models. If $\hat{d}(k)$ is the approximation of $d(k)$, the parallel NARMA model is described by

$$\hat{d}(k) = \hat{G}(\hat{d}(k-1), \hat{d}(k-2), \ldots, \hat{d}(k-n_d), u(k), \\ u(k-1), \ldots, u(k-n_u), \theta_k) \tag{4.5}$$

This model can predict the output given only the input. In contrast to this, the serie-parallel model uses the system outputs history to predict future

values. It is described by

$$\hat{d}(k) = \hat{G}(d(k-1), d(k-2), \ldots, d(k-n_d), u(k), \\ u(k-1), \ldots, u(k-n_u), \theta_k) \tag{4.6}$$

In many cases, the parameters θ_k may vary with time. An adaptive identification is then required to track these variations. Usually, gradient descent methods like recursive least square (RLS) and recursive prediction error (RPE) are widely used to adjust the model parameters.

4.3.2 Problem Statement

The problem to solve is that of developing an adaptive ESN identifier for nonlinear systems described by (4.4). The order and the nonlinear function of the system are assumed to be unknown. Furthermore, the system may have time-varying parameters. Upon completion of the training procedure, we expect that the trained ESN will be capable to exhibit characteristics, normally ascribed to adaptive identifiers: i) make an acceptable generalization to new incoming data, ii) detect the parameters variation; and iii) instantiate this information and begin making predictions in this new situation. Naturally, we only expect adequate performances for system conditions that are close to those seen during training.

4.3.3 Procedure

A question may be asked: Why the echo state property of the internal state can make this approach work in identifying systems (4.4)? Assume that we consider the serie-parallel model (4.6) for the identification. If the echo state property is satisfied, then after long time running, each state x_i can be mapped by input/output histories through a function called "echo function"[57]. Thus

$$x_i(n) = e_i(d(n-1), d(n-2), \ldots, u(n), u(n-1), \ldots) \tag{4.7}$$

where e_i are echo functions. Assume that there is no back-connection from the output to the "dynamic reservoir"(DR), and no synaptic weight connections from the input directly to the output, then the output of the ESN will be:

$$Y(n+1) = f^{out}(W^{out}(X(n+1))) \tag{4.8}$$

The trained network output is now a combination of the network states, which in turn are governed by these echo functions. To facilitate notation, we consider the case of a SISO system. In this case the network has only one output $y(n)$ given by

$$y(n+1) = f^{out}(\sum W_i^{out} e_i(d(n-1), d(n-2) \\ \ldots, u(n), u(n-1), \ldots)) \tag{4.9}$$

We can observe the connection between (4.4) and (4.9). The ESN approach propose to approximate (4.4) by a linear combination of many echo functions e_i in (4.9). The fact that the ESN has the "echo state property" gives e_i the same arguments as the function G in (4.4). Both are collection of inputs and outputs history.

4.3.4 Results

We performed several adaptive system identification, two examples taken from the literature are reported here. In all tests, we estimated the normalized mean square error defined as:

$$NMSE_{test} = \frac{E[(y_{test} - y)^2]}{\sigma^2} \tag{4.10}$$

where σ^2 is the variance of the system output y, and y_{test} is the ESN prediction.

Example 1

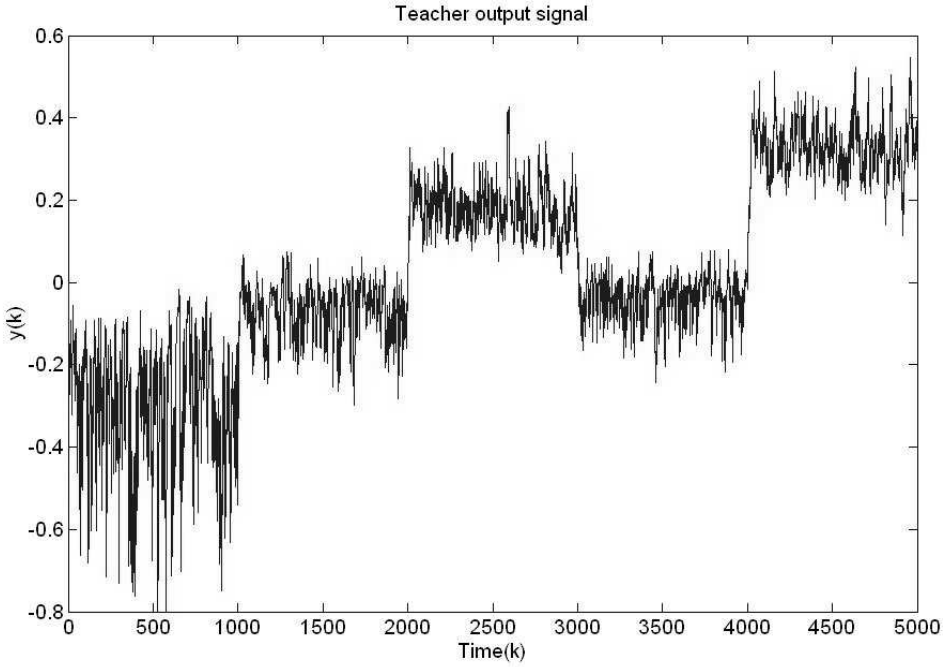
Let us consider the system described by the following difference equation [77]:

$$y(k+1) = \theta_1 y(k) + (u(k) - \theta_1)u(k)(u(k) + \theta_2) \tag{4.11}$$

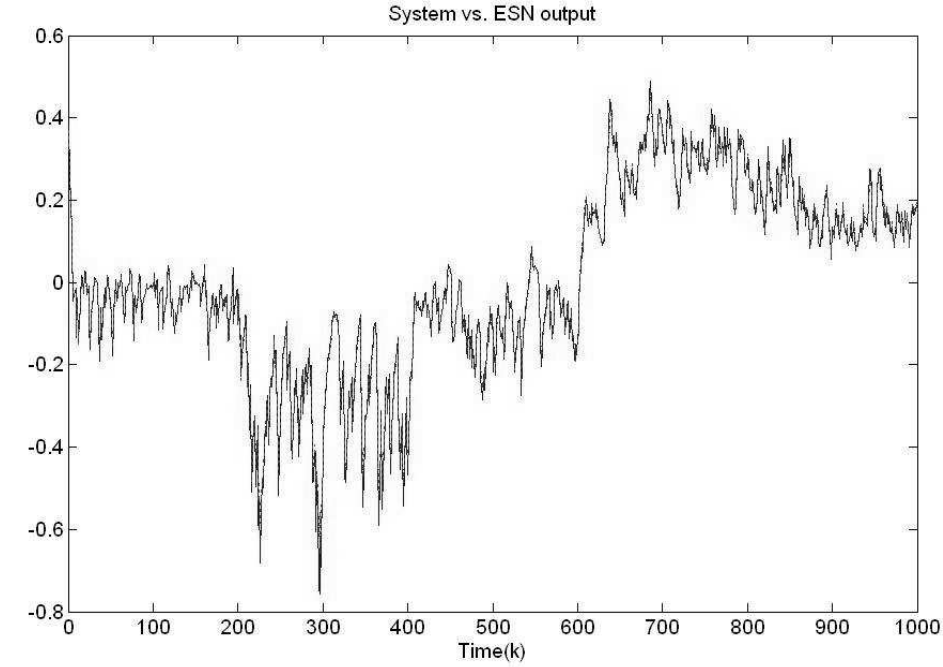
where $u(k)$ and $y(k)$ are the input and output of the system, respectively. Here, we assume that the parameters (θ_1, θ_2) have predefined possible values, given by the set $\Theta = \{(0.8, -3), (0.8, 0), (0.8, 3), (0.7, 0), (0.9, 3)\}$.

Input/Output (I/O) training data are prepared by driving the system (4.11) with an input sequence sampled from the uniform distribution over $[-2, +2]$. For each value of (θ_1, θ_2) from Θ, 1000 sequences are collected to form the off-line training set. As a result 5000 heterogeneous I/O sequences are used for training. Figure (4.6.a) shows the resulting scaled output teacher signal,

which clearly shows transitions every 1000 steps. Training was performed using an ESN with 2 inputs, which receive $(u(k), y(k-1))$, 9 internal neurons, and one output, which represents the predicted system output (y_{test}). No back-connection from the output to the DR, and no synaptic weight connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-0.5, +0.5]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eingenvalue $|\lambda_{max}| \approx 0.1$. After training, we used the same parameters seen during training, but in a different order. Inputs are sampled from a uniform distribution over $[-2, +2]$, then five sequences corresponding to
$(\theta_1, \theta_2) = \{(0.7, 0), (0.8, -3), (0.8, 0), (0.9, 3), (0.8, 3)\}$ are collected. Each sequence consists of 200 I/O test data. On test data, the ESN identifier showed a mean square error of $NMSE_{test} \approx 0.0226$. The network could make a reasonable generalization on new I/O data, and could recognize parameters variation only through its inputs and its state, without changing any synaptic weight. Figure(4.6.b) illustrates the obtained results.
In the next example, the performance of the ESN is tested on a higher order system. Tests will be done on new incoming data and new parameters variations.



(a)



(b)

Figure 4.6: Results of example 1. (a). Teacher output. (b). ESN Prediction test on new I/O data. Desired (solid) and network prediction (dashed) signals.

Example 2

Our goal here is to identify a 10th order nonlinear system, in a presence of time varying parameters. This example is taken from [83], where the system to be identified is given by the difference equation:

$$y(k+1) = 0.3y(k) + 0.05y(k)\Big[\sum_{i=1}^{9} y(k-i)\Big] + 1.5u(k-9)u(k) + 0.1 \tag{4.12}$$

In this thesis, we replace its fixed parameters by time varying parameters α, β, γ, and φ to obtain:

$$y(k+1) = \alpha y(k) + \beta y(k)\Big[\sum_{i=1}^{9} y(k-i)\Big] + \gamma u(k-9)u(k) + \varphi \tag{4.13}$$

I/O training data are prepared by driving the system (4.13) with a 13000-step input sequence sampled from the uniform distribution over $[0, 0.3]$. Every 1000 steps, α,β,γ, and φ were assigned new random values taken from $\pm 50\%$ interval around their nominal values. The first 10000-steps I/O are used for training, and the other 3000-steps for generalization tests. Figure (4.7.a) shows the resulting output teacher signal, which also shows transitions every 1000 steps.
For problems with long-term time dependencies, RNNs are very hard to train [47]. The system treated here has a time-lag of ten time steps, and using the training input pair $(u(k), y(k-1))$ was not enough to give complete information about the dynamics of the system, resulting to poor performances of the ESN identifier. Introducing one more delay from the system $(u(k), y(k-1), y(k-2))$ as inputs allowed the ESN identifier to demonstrate better performances. Therefore, the ESN architecture was chosen as follows. 3 inputs, 13 internal neurons, and one output. No back-connection from the output to the DR, and no synaptic weight connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-0.5, +0.5]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eingenvalue $|\lambda_{max}| \approx 0.3$. On test data, the network showed a mean square error of $NMSE_{test} \approx 0.0066$. Results are illustrated in Figure(4.7. b).

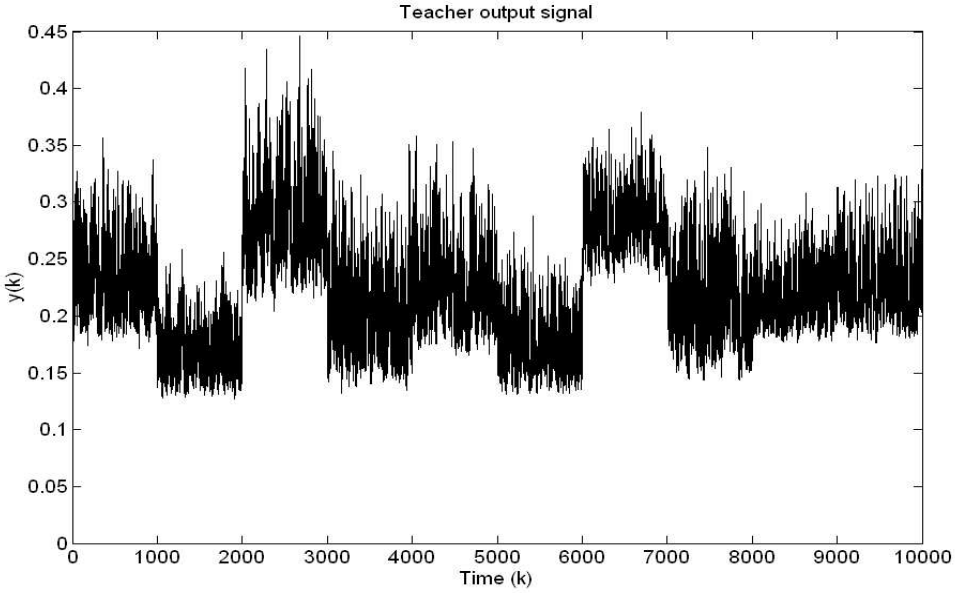
We performed then two other tests using two different inputs collected from (4.14) and (4.15), and new random parameters listed on Table 1. On test data obtained from the inputs u_1 and u_2, the ESN identifier showed a mean square error of $NMSE_{test} \approx 0.0058$ and 0.0052, respectively. Results of the two tests are illustrated by Figure(4.8. (a) and (b)).

Table 4.1: System parameters values and their time intervals

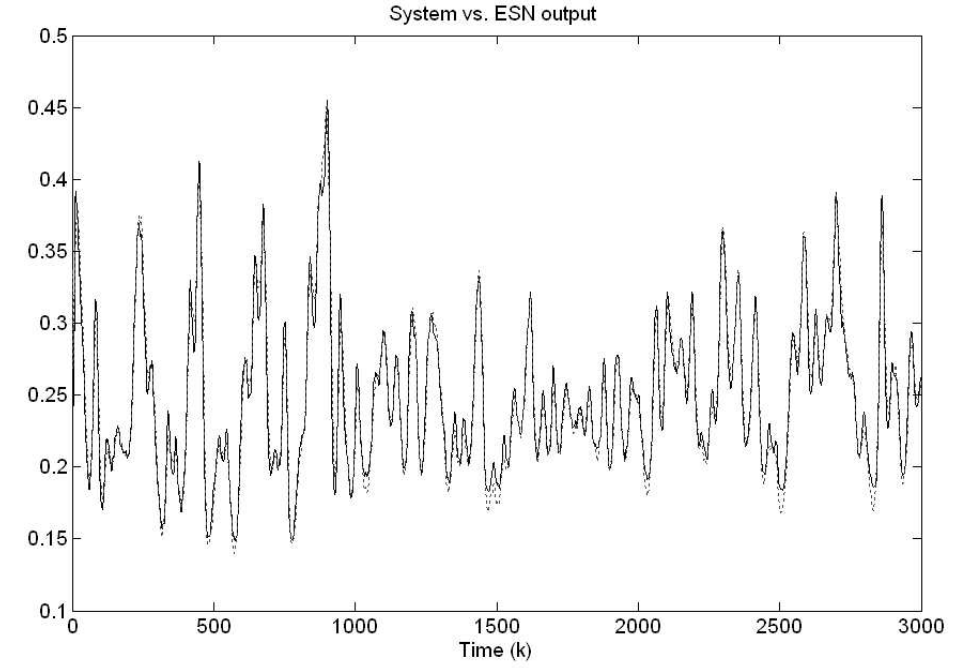
Time	α	β	γ	φ
$1 \leq k \leq 1000$	0.317	0.041	1.558	0.108
$1001 \leq k \leq 2000$	0.318	0.039	1.725	0.077
$2001 \leq k \leq 3000$	0.329	0.058	1.475	0.089
$3001 \leq k \leq 4000$	0.356	0.051	1.780	0.083

$$u_1(k) = 0.05\Big[\sin(\frac{\pi k}{30}) + 2.7\Big] + 0.02\Big[\sin(\frac{\pi k}{10}) + 2\Big] \tag{4.14}$$

$$u_2(k) = 0.05\Big[sign(0.1\cos(\frac{\pi k}{150}) + 3.5)\Big] \tag{4.15}$$

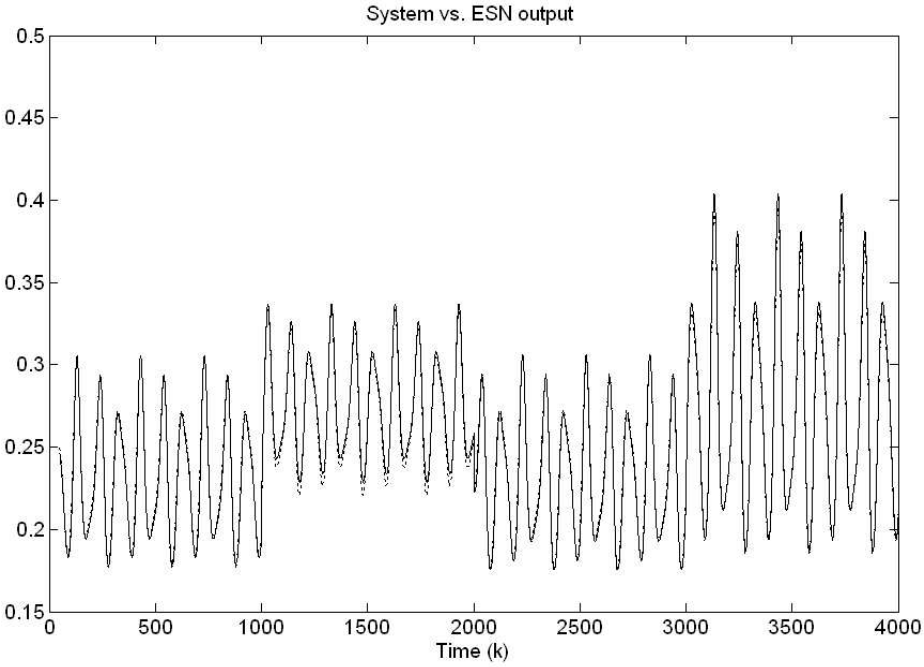


(a)

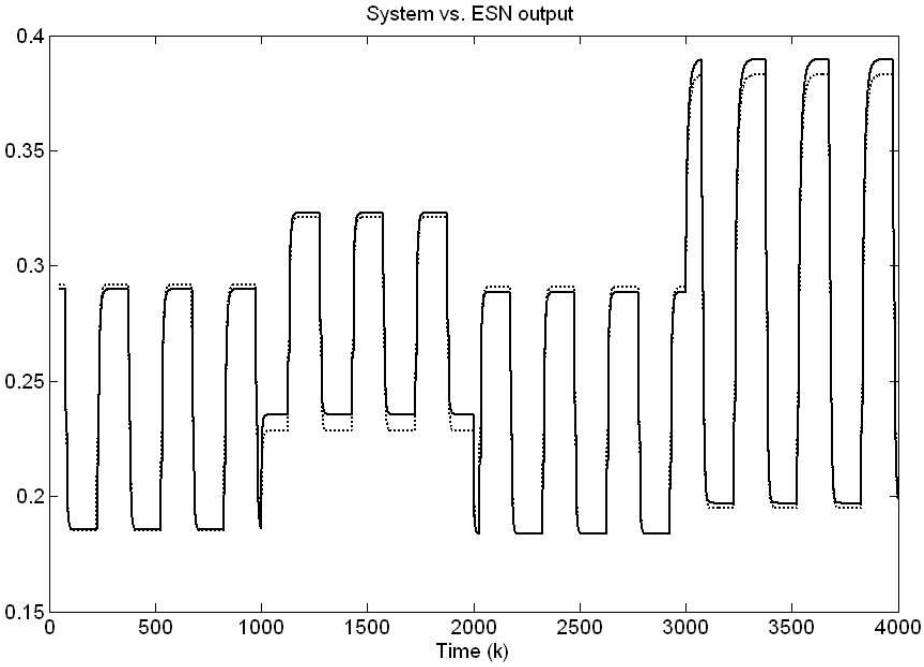


(b)

Figure 4.7: Results of example 2 (1st test). (a). Teacher output. (b). ESN Prediction test on new I/O data. Desired (solid) and network prediction (dashed) signals.



(a)



(b)

Figure 4.8: Results of example 2 (2nd test). System(solid) vs ESN Prediction(dashed). (a). Responses to $u_1(k)$. (b). Responses to $u_2(k)$.

4.4 Conclusion

Like any survey, this chapter did not cover many other relevant works conducted on Metalearning. The objective was to make the reader familiar with this concept, which is central in this thesis. Despite the different meaning ascribed to metalearning, no clear consensus exists of what is meant by this term. However, the question to solve remains constant: *How can we improve learning algorithms by exploiting past knowledge (learning experience)*? So a possible unification of current efforts in this promising area could be resumed as the building of self-adaptive learners that improve their bias dynamically according to the task at hand [62].
Several researchers have used metalearning to build up neural networks that have the ability to adapt without changing explicit weights. Two directions are used in the literature. The first one focuses on the concept of learning to implement a learning rule, the second, instead, is built on the concept of training a RNN that would approximate different families of functions. This thesis follows the second direction to develop fixed-weight adaptive identifiers/controllers based on ESNs. In the simulations, an ESN is asked to identify several nonlinear dynamical systems with time varying parameters. In other words, the ESN is being asked to exhibit a characteristic, normally ascribed to adaptive systems, whose parameters change in response to an environmental change. In such problems, neural networks are usually trained on-line to contineously update their weights. This method is not completely satisfactory, due to several problems, i.e local minimum, slow convergence and complexity of the learning process. The fixed-weight solution adopted here presents a clever and efficient strategy compared with adaptive networks. "Adaptation" in this work is defined as the ability of the network identifier to change its behavior policy only by switching its internal state, represented by recurrence connections, without changing any synaptic weight. This capability is a natural consequence of prior training. The obtained results indeed confirm the ability of the fixed-weight ESN identifier to handle balance between the variety of new I/O data and the variety of the parameters. Furthermore, no multi-streaming is needed during training, since ESNs use a batch-learning algorithm and don't suffer from the recency effect.

The results of the first example compare favourably to the results for the same problem presented in [77]. Furthermore, the ESN solved not only the same problem presented in [83], but also its general form where parameters are replaced with time varying ones.

In this chapter, the examples highlighted the universal approximation capability of the ESNs. The two following chapters discuss, how ESNs can be used in mobile robot control.

Chapter 5

Control of Nonholonomic Robots with ESNs

In this chapter, an adaptive neurocontrol system with two levels is proposed for the motion control of a nonholonomic mobile robot. In the first level, an ESN improves the robustness of a kinematic controller and generates linear and angular velocities, necessary to track a reference trajectory. In the second level, another ESN network converts the desired velocities, provided by the first level, into a torque control. The advantage of the control approach is that, no knowledge about the dynamic model is required, and no synaptic weight changing is needed in presence of robot's parameters variation. Furthermore, we demonstrate the ability of a single fixed-weight ESN to act as a dynamic controller for several (here 3) distinct mobile robots. This capability is acquired through prior 'metalearning'. Simulation results are demonstrated to validate the robustness of the proposed approach.

5.1 Introduction

Control of mobile robots has been studied by many authors in the last decade, since they are increasingly used in wide range of applications. At the begin-

ning, the research effort was focused only on the kinematic model, assuming that there is *perfect* velocity tracking [28]. Later on, the research has been conducted to design navigation controllers, including also the dynamics of the robot [29][33]. Taking into account the specific robot dynamics is more realistic, because the assumption "perfect velocity tracking" does not hold in practice. Therefore, it is desirable to develop a robust motion control, which has the following capabilities: i) ability to successfully handle estimation errors and noise in sensor signals, ii) velocity tracking at the dynamic-level, and iii) adaptation ability, in presence of time varying parameters.

As seen in the previous chapter, we trained an ESN to act as an adaptive identifier of non linear dynamical systems, with fixed weights [82]. In this chapter, a control approach based on ESNs is developped for tracking control of a nonholonomic mobile robot. Two control levels are designed. First, an ESN kinematic controller provides the desired velocity, necessary to minimize the position error. Then, an adaptive dynamic ESN controller transforms the desired velocity into torques for wheels, such that the robot's velocity converge to the desired one. In this approach, no dynamical model is required and no synaptic weight changing is needed in presence of parameters variation.

The rest of this chapter is organized as follows. The two next sections provide a review of modelling and control of nonholonomic mobile robots. The design of the adaptive dynamic ESN controller is described in sections (5.4) and (5.5). The problem of developping a single controller for three distinct robots is discussed in section (5.6). In Section (5.7), the neurocontrol system with two levels is designed for motion control, and some simulation results are presented. Finally, dicussions and conclusion are drawn in sections (5.8) and (5.9).

5.2 Nonholonomic Mobile Robots

In this section, a review of modeling and control of Nonholonomic mobile robots is provided. In such robots, the motion control will be subject to nonholonomic constraints, which makes motion perpendicular to the wheels impossible. This constraint involves a non trivial control methods although the full state is measured.

5.2.1 Kinematic and Dynamic Modeling

A mobile robot system having an n-dimensional configuration space with generalized variables $(q_1, q_2, ..., q_n)$ and subject to constraints can be described by [33]:

$$M(q)\ddot{q} + V_m(q, \dot{q})\dot{q} + F(\dot{q}) + \tau d = B(q)\tau - A^T(q)\lambda \tag{5.1}$$

where $M(q) \in \Re^{n\times n}$ is a symmetric, positive definite inertia Matrix, $V_m(q, \dot{q}) \in \Re^{n\times n}$ is the centripetal and coriolis matrix, $F(\dot{q})$ denotes the surface friction, τ_d denotes bounded unknown disturbances including unstructured unmodeled dynamics, $B(q) \in \Re^{n\times r}$ is the input transformation matrix, is the input vector, is the matrix associated with the constraints, and $\lambda \in \Re^{m\times 1}$ is the vector of constraint forces. The nonholonomic nature of a mobile robot is related to the assumption that the wheels of the vehicle roll without slipping. They are subject to non-integrable equality nonholonomic constraints involving the velocity. In other words, the dimension of the admissible velocity space is smaller than the dimension of the configuration space. This constraint can be written:

$$A(q)\dot{q} = 0 \tag{5.2}$$

In the case of a mobile Robot with two actuated wheels, I will adopt the model used in [29]. Consider the mobile robot of Figure (5.1). O_xy is the world coordinate system and P_0_XY is the coordinate system fixed to the mobile robot. P_0 is the origin of the coordinate system P_0_XY and the middle between the right and left driving wheels. The distance from P_0 to the center of mass P_c is d.

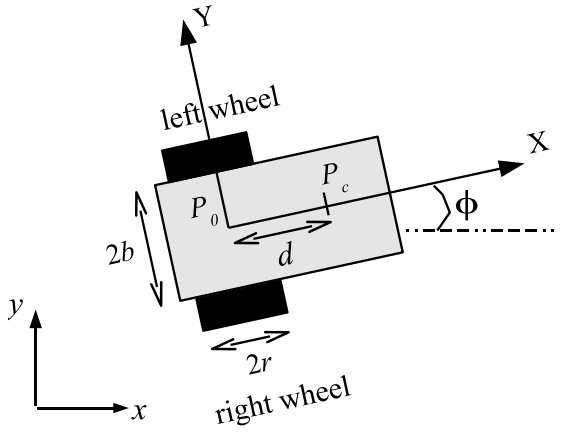


Figure 5.1: Mobile robot with two actuated wheels.

For the later description, m_c and m_w are the mass of the body and wheel with a motor, I_c, I_w, and I_m are the moment of inertia of the body about the vertical axis through P_c, the wheel with a motor about the wheel axis, and the wheel with a motor about the wheel diameter, respectively.

The matrices M,V and B are given by :

$$M(q) = \begin{bmatrix} m & 0 & 2m_w d\sin\phi & 0 & 0 \\ 0 & m & -2m_w d\cos\phi & 0 & 0 \\ 2m_w d\sin\phi & -2m_w d\cos\phi & I & 0 & 0 \\ 0 & 0 & 0 & I_w & 0 \\ 0 & 0 & 0 & 0 & I_w \end{bmatrix}$$

$$V(q,\dot{q}) = \begin{bmatrix} 2m_w d\dot{\phi}^2\cos\phi \\ 2m_w d\dot{\phi}^2\cos\phi \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; B(q) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}, \; \tau = \begin{bmatrix} \tau_r \\ \tau_l \end{bmatrix}$$

I and m are given by: $I = m_c d^2 + I_c + 2m_w b^2 + 2I_m$, $m = m_c + 2m_w$.

Five generalized coordinates can describe the configuration of the mobile robot: $q = [x, y, \phi, \theta_r, \theta_l]^T$ where (x, y) are the coordinates of P_0, ϕ is the heading angle of the mobile robot, and θ_r, θ_l are the angles of the right and left driving wheels.

We assume the wheels roll and do not slip. Then, there exist three constraints; the velocity of P_0 must be in the direction of the axis of symmetry and the wheels must not slip:

$$\dot{y}\cos\phi - \dot{x}\sin\phi = 0 \tag{5.3}$$

$$\dot{x}\cos\phi + \dot{y}\sin\phi + b\dot{\phi} = r\dot{\theta}_r \tag{5.4}$$

$$\dot{x}\cos\phi + \dot{y}\sin\phi - b\dot{\phi} = r\dot{\theta}_l \tag{5.5}$$

From the constraints (5.3), (5.4) and (5.5), let a matrix A be defined as:

$$A(q) = \begin{bmatrix} \sin\phi & -\cos\phi & 0 & 0 & 0 \\ \cos\phi & \sin\phi & b & -r & 0 \\ \cos\phi & \sin\phi & -b & 0 & -r \end{bmatrix} \tag{5.6}$$

Let $S(q)$ be a full rank matrix formed by a set of smooth and linearly independent vector such as:

$$S^T(q)A^T(q) = 0 \tag{5.7}$$

It is easy to verify that $S(q)$ is given by:

$$S(q) = \begin{bmatrix} \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ \frac{r}{2b} & -\frac{r}{2b} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.8}$$

According to (5.1) and (5.7), it is possible to find that:

$$\dot{q} = S(q)\nu \tag{5.9}$$

where $\nu = [\nu_1, \nu_2]$ are the angular velocities of the right and left wheels. Equation (5.9) represents the kinematic model of the robot.

Differentiating (5.9), substituting the result in (5.1), and then multiplying by S^T, we can eliminate the constraint matrix $A^T(q)\lambda$. Also, if we denote $\bar{M} = S^T M S$ and $\bar{V} = S^T(MS + VS)$, and after simplifications, the nonholonomic mobile robot (5.1) can be transformed to the following equations:

$$\bar{M}(q)\dot{\nu} + \bar{V}(q,\dot{q})\nu = \bar{B}(q)\tau \tag{5.10}$$

where

$$\bar{M}(q) = \begin{bmatrix} \frac{r^2}{4b^2}(mb^2 + I) + I_w & \frac{r^2}{4b^2}(mb^2 - I) \\ \frac{r^2}{4b^2}(mb^2 - I) & \frac{r^2}{4b^2}(mb^2 + I) + I_w \end{bmatrix} \tag{5.11}$$

$$\bar{V}(q) = \begin{bmatrix} 0 & \frac{r^2}{2b} m_c d\dot{\phi} \\ -\frac{r^2}{2b} m_c d\dot{\phi} & 0 \end{bmatrix} \tag{5.12}$$

$$\bar{B} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.13}$$

and $\tau = [\tau_r, \tau l]^T$ consists of motors torques τ_r and τ_l , which act on the right and left wheels, respectively.

Equations (5.9) and (5.10) represent the kinematic and dynamic models of the robot, respectively.

From equation (5.9) we obtain:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \phi \\ \theta_r \\ \theta_l \end{bmatrix} = \begin{bmatrix} \frac{r}{2}\cos\phi & \frac{r}{2}\cos\phi \\ \frac{r}{2}\sin\phi & \frac{r}{2}\sin\phi \\ \frac{r}{2b} & \frac{-r}{2b} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} \tag{5.14}$$

The relation between (v, w) and (ν_1, ν_2) is:

$$\begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{r} & \frac{b}{r} \\ \frac{1}{r} & \frac{-b}{r} \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{5.15}$$

where v and w are the linear and angular velocity of the robot. If we want to focus only on x, y, and ϕ then it is sufficient to substitute (5.15) for (5.14). We will get the ordinary form of a mobile robot with two actuated wheels:

$$\frac{d}{dt}\begin{bmatrix} x \\ y \\ \phi \end{bmatrix} = \begin{bmatrix} \cos(\phi) & 0 \\ \sin(\phi) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} \tag{5.16}$$

5.3 Motion control

Motion control of mobile robots has been studied by many authors in the last decade, since they are increasingly used in wide range of applications. In motion control, the objective is to control the velocity of the robot such that its pose $P = [x, y, \phi]^T$ follows a reference trajectory $P_r = [x_r, y_r, \phi_r]^T$. At the beginning, the research effort was focused only on the kinematic model, assuming that there is *perfect velocity tracking*. Thus, the controllers neglect the vehicle dynamics and consider only the steering system (5.16), where the velocities (v,w) are supposed to be the robot inputs (Figure 5.2). Various feedback controllers have been proposed to solve this problem (see the survey paper [28] and references cited therein).

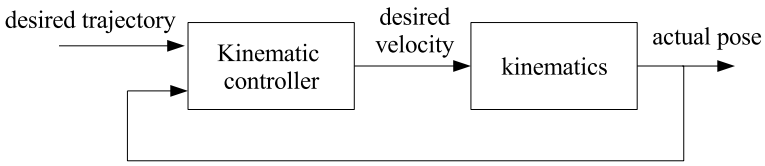


Figure 5.2: Motion control using the kinematic model.

Later on, the research has been conducted to design motion controllers, including also the dynamics of the robot. Taking into account the specific robot dynamics is more realistic, because the assumption "perfect velocity tracking" does not hold in practice. In a real case, the mobile robot is described with two stages. As mentioned in Figure (5.3), the first stage contains the dynamics, with two outputs (v, w) linear and angular velocity. The second stage contains the kinematics.

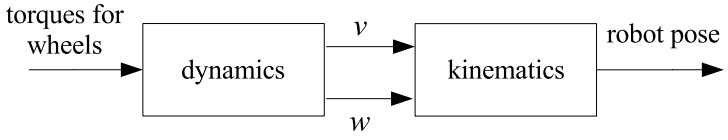


Figure 5.3: Two-stage model of a real mobile robot.

To have better motion control performance one has to take into account also the specific vehicle dynamics. In this case the controller structure should be decomposed into two stages:

1. an inner loop (Figure 5.4), depending on the robot dynamics, and can be used for controlling the linear and angular velocity. It is also called *dynamic-level* control of a mobile robot.

2. an outer loop, controls the pose of the robot. It is also called *kinematic-level* control of a mobile robot.

   The global control structure is then decomposed as presented in Figure(5.5).

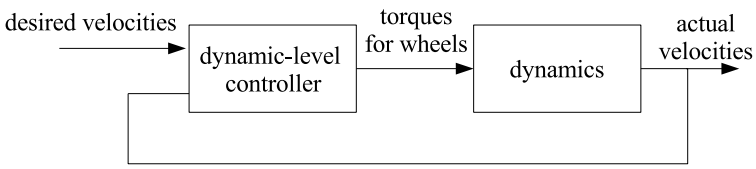


Figure 5.4: Inner loop control of a mobile robot (dynamic-level control).

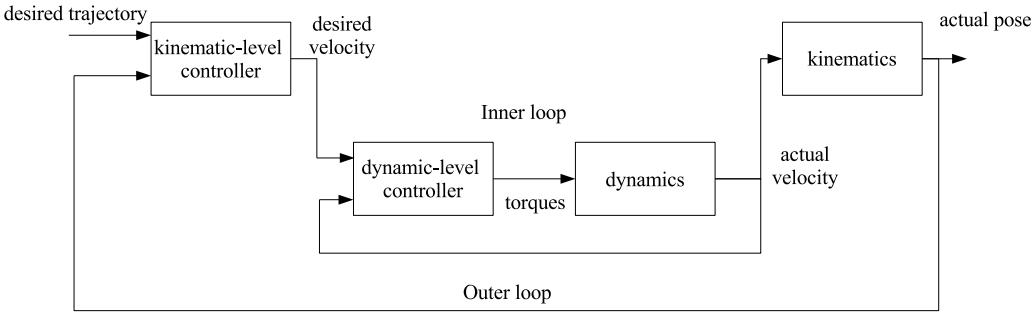


Figure 5.5: Control of a real mobile robot.

Little has been written about integrating the kinematic controller and the dynamics of the mobile robot, and the literature on robustness and control in presence of uncertainties in the dynamical model of such systems is rare [84]. However, when the dynamic model is considered, exact knowledge about the parameters values of the mobile robot dynamics is almost unattainable in practical situations. If we consider that these parameters are time varying, the problem becomes more complicated. At present, PID controller is still widely used in motor control of mobile robot. However, its ability to cope with some complex process properties such as nonlinearities,

and time-varying parameters are known to be very poor. In the next sections, an adaptive neurocontrol system with two levels is proposed for the motion control of a nonholonomic mobile robot. In the first level, an ESN improves the robustness of a kinematic controller and generates linear and angular velocities, necessary to track a reference trajectory. In the second level, another ESN converts the desired velocities, provided by the first level, into a torque control. The advantage of the control approach is that, no knowledge about the dynamic model is required, and no synaptic weight changing is needed in presence of robot's parameters variation.

5.4 Dynamic-level Control with ESNs

In this section, the ESN is asked to act as a dynamic-level controller for nonholonomic mobile robots. It is desirable to consider that in practical situations, the knowledge about the dynamic model is not complete.

5.4.1 Procedure

The advantage of the control approach proposed is that no knowledge about the dynamic model of the robot is required. This property makes it very useful in practical situation, where the exact knowledge about the real parameters is almost unattainable. Here, the ESN dynamic controller (we call it ESN_D) is designed only by learning I/O data collected from the dynamic model (5.10).

Training

Training procedure involves using actual and delayed velocities of the robot as inputs, and correspondent torques as teacher signals (Figure 5.6). The goal of this training procedure can be summarized as follow: Find the weights of the network using the teacher signal, which brings the robot from the actual velocity at time (n) to a future velocity at time $(n+1)$[85].

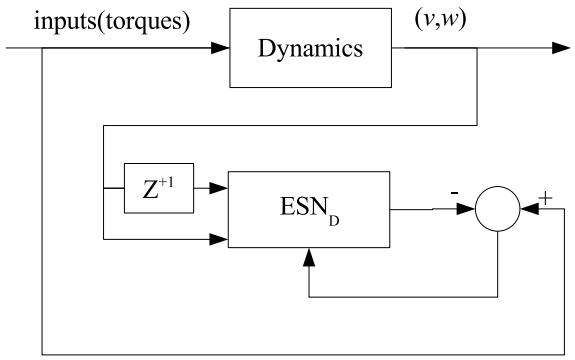


Figure 5.6: Training ESN_D as a dynamic controller.

Exploitation

After training procedure, the ESN_D controller is attached to the robot as low-level closed loop controller (Figure 5.7). The ESN_D used the actual (w, v) and the desired (w_d, v_d) angular and linear velocities, to give the appropriate control inputs (torques).

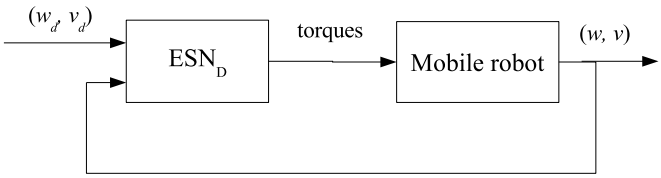


Figure 5.7: Exploitation of ESN_D as a dynamic controller.

5.4.2 Results

In the simulation, the robot's nominal parameters values are chosen as follows [29]: $a = 2, b = 0.75, d = 0.3, r = 0.15, m_c = 30, m_w = 1, I_c = 15.625, I_w = 0.005, I_m = 0.0025$. The network used in these simulations has 4 inputs (actual and delayed linear and angular velocities of the robot), 2 outputs (two torques for robot dynamical model), no back-connection from the output to the dynamic reservoir (DR), and no synaptic weight connections from the input directly to the output. The input and the internal synaptic connections weights are randomly initialized from a uniform distribution over $[-1, +1]$.

To train the ESN_D, we used collected Data from the model (5.10). First, random inputs (torques) are generated. Then the robot is driven using these inputs with an open loop, in order to collect the correspondent angular and linear velocities. 4000 data were collected: 2000 data used to train the network and the 2000 others are used for generalization tests. The dimension of the DR and the maximum eigenvalue were tuned by hand, in order to avoid overfitting, and have a good generalization. After many tests, we found that the best DR dimension is between 18 and 25 units, and an optimum $|\lambda_{max}|$ is between $0,3$ and $0,6$. In the simulation, 20 internal units are used, and the internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eigenvalue $|\lambda_{max}| \approx 0.4$.

After training and generalization tests, the performances of the ESN_D controller were tested on new desired linear and angular velocities:

$$
\begin{aligned}
1 \leq t \leq 100 : v_r &= 0.25(1 - \cos \tfrac{\pi t}{5}) \\
w_r &= -v_r
\end{aligned}
$$

$$
\begin{aligned}
100 \leq t \leq 200 : v_r &= 0.5 \\
w_r &= -0.5
\end{aligned}
$$

$$
\begin{aligned}
200 \leq t \leq 300 : v_r &= 0.5 \\
w_r &= 0.5(1 - \cos \tfrac{\pi t}{5}) - 0.5
\end{aligned}
$$

$$
\begin{aligned}
300 \leq t \leq 700 : v_r &= 0.5) \\
w_r &= 0.5
\end{aligned}
$$

$$
\begin{aligned}
700 \leq t \leq 800 : v_r &= 0.5 \\
w_r &= 0.25(1 - \cos \tfrac{\pi t}{5})
\end{aligned}
$$

$$
\begin{aligned}
800 \leq t \leq 900 : v_r &= 0.25\pi(1 + \cos \tfrac{\pi t}{5}) \\
w_r &= 0
\end{aligned}
$$

Figure (5.8) shows the obtained results. As can be seen, the robot model tracks simultaneously and perfectly the desired linear and angular velocity. The mean square error between actual and desired linear velocities is $MSE_{Linvelo} = 5.7670e - 007$ and between the actual and desired angular velocities is $MSE_{Angvelo} = 1.2057e - 005$. This result demonstrates that the ESN approach is able to control the dynamics of the robot, without prior

information about the dynamic model. This property makes it very useful in practical situation, where the exact knowledge about the mobile robot parameters is almost unattainable.

Next section will discuss the ability of the ESN controller to adapt with fixed-weight.

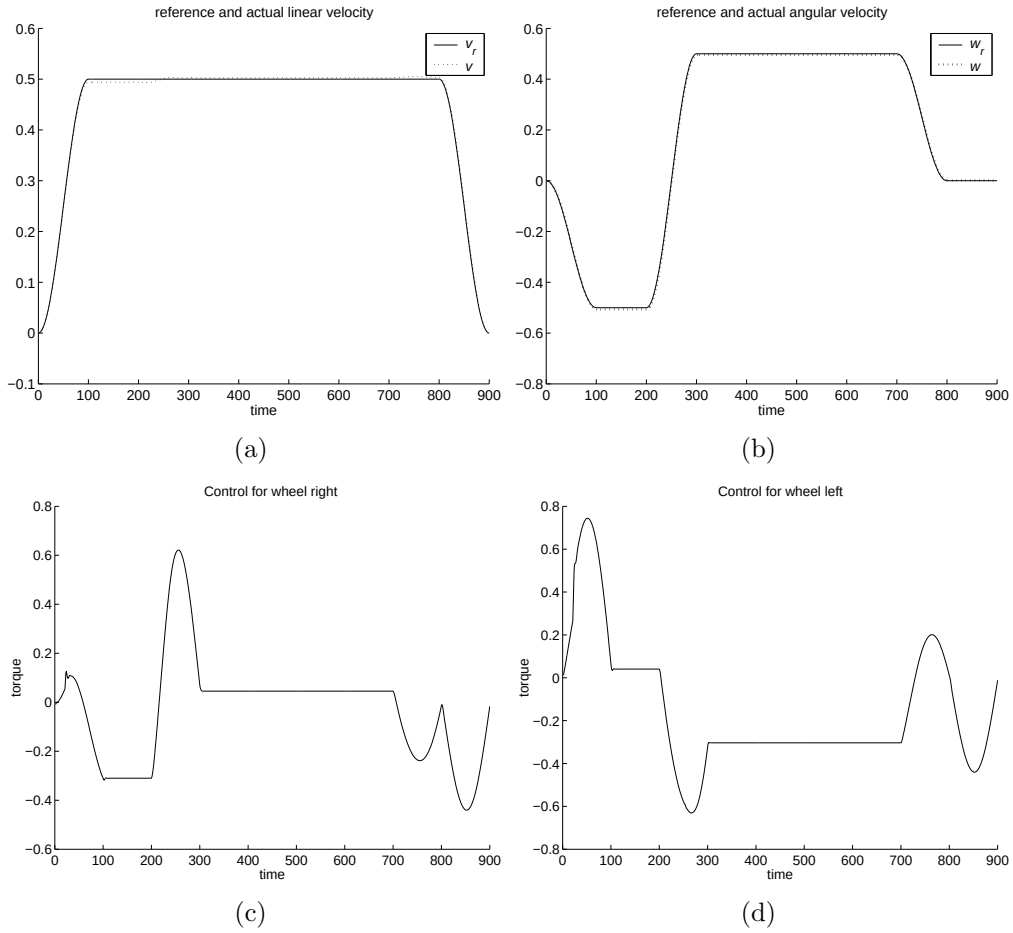


Figure 5.8: ESN adaptive velocity tracking control. a) Linear velocity tracking. b) Angular velocity tracking. c) Computed torque for wheel right. d) Computed torque for wheel left.

5.5 Adaptive Dynamic Control using Fixed-Weight ESNs

Training an echo state network as a dynamic controller for nonholonomic mobile robots is developed in section (5.4). Here, however, we are interested in much more interesting problem of requiring a fixed-weights ESN to make an adaptive dynamic control. In chapter (3), we explored the notion that a well trained ESN needs to change only its internal state to change its behavior policy. It was shown how metalearning provides the ESN identifier the ability to adapt only by switching its internal state, represented by recurrence connections, without changing any synaptic weight. In this section, metalearning will be used to build up an adaptive ESN_D controller, in presence of time varying parameters. Furthermore, this ability allows a single fixed weight ESN_D to act as a dynamic controller for several distinct robots.

5.5.1 Problem Statement

In this section, an ESN dynamic controller is designed for the model (5.10), in presence of time varying parameters (d, m_c). We assume that m_c vary between the values $[30, 35]$ and the distance d can vary in interval of $\pm 50\%$ around its nominal value. Upon completion of training, we expect that the ESN dynamic controller will be capable to track reasonably new (not seen during training) predefined linear and angular velocities, and to exhibit characteristics, normally ascribed to adaptive controllers: i) detect the variation of the mass m_c and the distance d; ii) adapt itself to the perceived change; and iii) generate the appropriate control signal, without changing any synaptic weight.

5.5.2 Procedure

Here, we have a typical case, where adaptive control techniques are needed. In the sense of control theory, an adaptive controller is an "intelligent" controller that can modify its behavior in response to the variations in the dynamics of the process and the character of the disturbances. This can be approached in different ways; Self-tuning control, and model reference adaptive control are the most popular techniques. Neural Networks can also give complementary/new solutions in adaptive control of complex non-linear dy-

namical systems. Adaptive behavior in neural networks can be achieved by finding specific learning mechanisms that are able to adapt a network to new or changed situations. One way to achieve this is to update the weights to the changing environement. This method is widely used in literature. Multilayer perceptrons (MLPs) with long-and short- term memory have been proposed for adaptive systems [77].
In this work, we propose to use the metalearning procedure to develop an adaptive velocity controller. As mentioned in chapter (3), metalearning can offer ESNs an adaptive behavior with fixed weights. Metalearning in this case is to train ESN for "all" possible operating conditions of the dynamical model, corresponding to possible combinations of the parameters values (d, m_c). I/O training data are prepared by driving (5.10) with input sequences sampled from the uniform distribution over $[-2, +2]$. Training was carried out using 10 blocs of 1000 sequences. In each bloc, m_c and d were assigned new random values taken from their respective variation intervals. In this work, no multi-stream is needed, since the ESN batch update uses all data at once, and does not suffer from the recency effect. To train ESN as a velocity controller, we used the same training approach presented in figure(5.6). The network architecture was chosen as follows. 4 inputs, which represent actual (v, w) and desired (v_d, w_d) linear and angular velocities, 30 internal neurons, and 2 outputs. No back-connection from the output to the DR, and no synaptic weight connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-1, +1]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eigenvalue $|\lambda_{max}| \approx 0.3$.

5.5.3 Results

After training, the ESN_D showed on test data a mean square error of $MSE = 1.352e-004$. To test the performance of the trained controller ESN_D, the parameters (m_c, d) are varied according to Table 5.1, and new predefined reference linear and angular velocities (v_r, w_r) are chosen as in section (5.4). Figures (5.9. (c) and (d)) show clearly the adaptation performance of the fixed weights ESN_D to robot parameters variations, and its generalization capability to track a new reference velocity.

Table 5.1: Values of m_c and d in their time intervals

Time	m_c	d
$0 \leq t \leq 200$	30	0.3
$200 \leq t \leq 300$	32	0.4
$300 \leq t \leq 500$	35	0.2
$500 \leq t \leq 700$	30	0.35
$700 \leq t \leq 800$	32	0.25
$800 \leq t \leq 900$	35	0.3

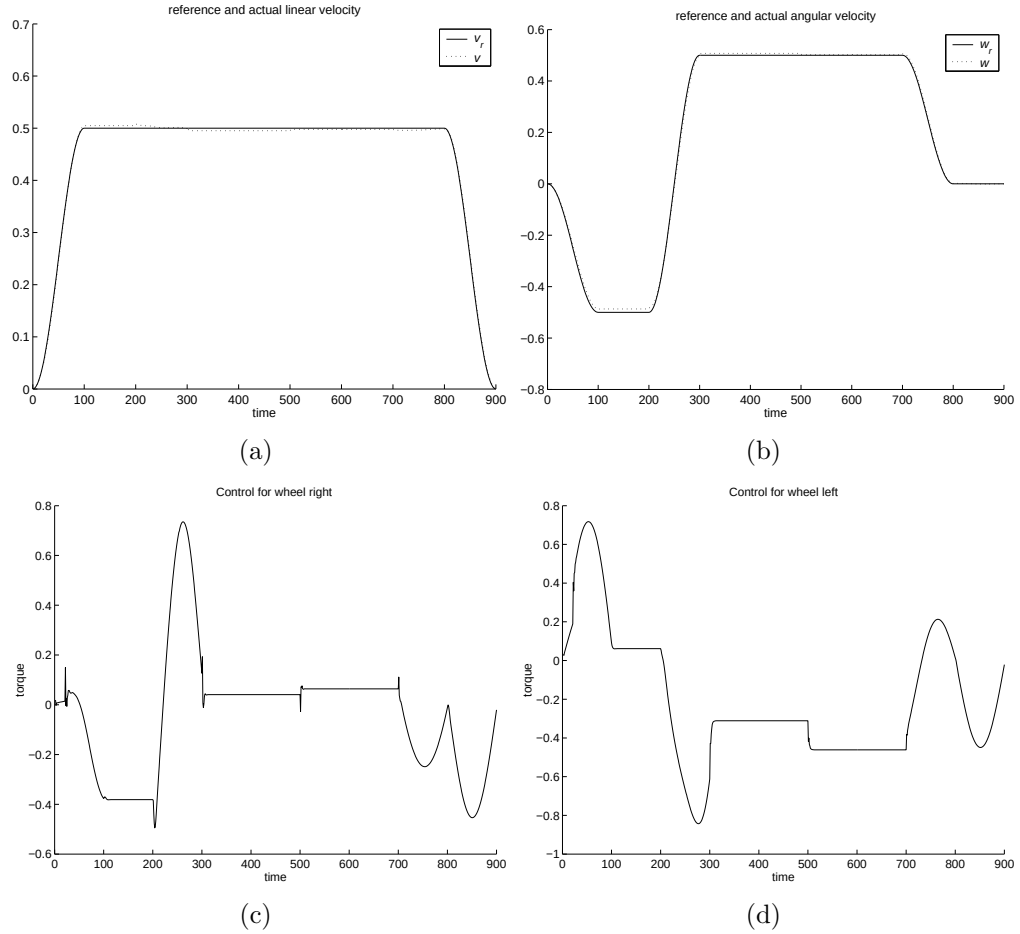


Figure 5.9: ESN adaptive velocity tracking control. a) Linear velocity tracking. b) Angular velocity tracking. c) Computed torque for wheel right. d) Computed torque for wheel left.

5.6 Control of Multiple Distinct Robots

In this paragraph, the problem to solve is that of developing a single ESN dynamic controller for three distinct nonholonomic mobile robots [86]. Each robot has the same model structure (5.10), but their specific parameters result in quite different behavior. Upon completion of the training, we expect that the ESN controller will be capable of detecting the identity of the robot, only from its input in combination with its own state, without changing any synaptic weight. Furthermore, we also desire that the trained network be capable to minimize reasonably errors between the reference and the robots velocities. In this work, the width b, the distance d and the wheel radius r are chosen to be different and specific for each robot. Table 5.2 lists their values, where we label the robots by I, II, and III. The other parameters values are the same for all robots and chosen as in section (5.4).

Table 5.2: Robots Specifications

	b	d	r
Robot I	0.4	0.1	0.05
Robot II	0.75	0.3	0.15
Robot III	0.3	0.2	0.25

5.6.1 Procedure

Training procedure uses the same training approach of figure (5.6). Training was carried out using 3000 random heterogeneous input-output sequences. Each 1000 sequences are collected from one robot. After many hand tuning, the ESN architecture was chosen as follows. 4 inputs (actual and reference linear and angular velocities of the robot), 17 internal neurons, 2 outputs (two torques), no back-connection from the output to the DR, and no synaptic weight connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-1, +1]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eigenvalue $|\lambda_{max}| \approx 0.3$.

5.6.2 Results

In the simulation, we performed several tests on the behavior of the trained network, two of which are reported here. In the first, the fixed-weight ESN controller was used to control each robot separately. In the second, we tested its capability to handle switch between the three robots. In the two cases, the linear and angular velocities to be followed are the same as in [29]. Control results for the first case are present in Figure (5.10) and those for the second case in figure (5.11). In each figure, the panels on the left-hand side show the evolution of a robot state (linear and angular velocities), while the right hand side panels provide the corresponding control signals (torques) given by the ESN controller. In figure (5.10), the right-hand column provides the three control signals delivered by the same fixed-weight ESN controller for the three robots separately. The three signals are superposed in order to facilitate the comparison, and to have an idea about the three robots behaviors. As can be seen on the left-hand panels, the excellent velocity tracking of all robots is evident. On the predefined reference linear and angular velocities, the robots I, II, and III showed mean square errors of $MSE = \{(1.7534e - 004, 1.2619e - 004), (7.9237e - 005, 8.6139e - 005), (0.0013, 7.3374e - 004)\}$, respectively. Here, the fixed-weight controller did a reasonable job, and could effectively deliver the appropriate control signals for each robot. The second case is more complicated. In this case, the controller had to handle balance between tracking the reference velocity and switch between the three robots. In figure (5.11), the first switch occurs at time $25s$ from Robot II to Robot I, and the second one occurs at time $35s$ from Robot I to Robot III. A switch from one Robot to another requires the controller outputs to change, since each robot has its proper dynamic characteristics. Surprisingly, the control is barely affected by these switches. The resulting network controller showed smooth and rapid adaptation to these changes (see right-hand panels), and the three robots tracked reasonably the reference linear and angular velocities in their respective time intervals.

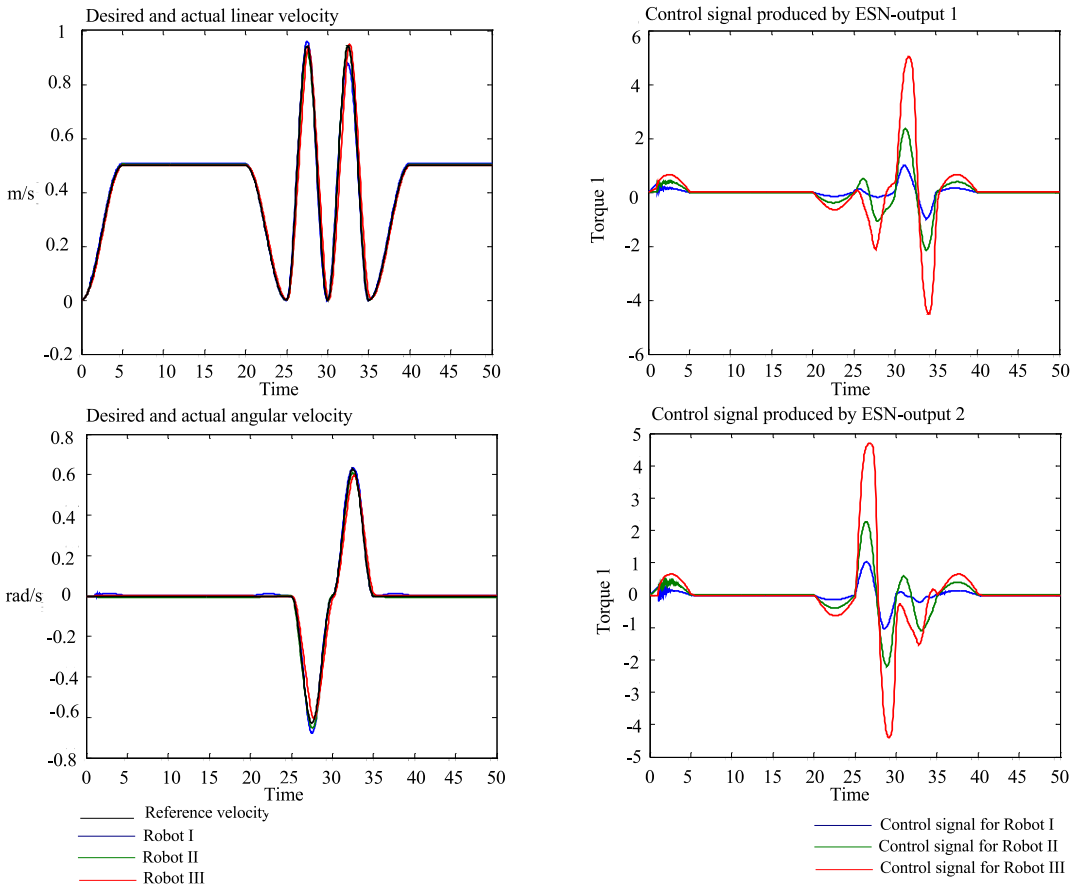


Figure 5.10: Linear and Angular velocity tracking (left) and controls (right). Each robot is controlled separately with the same fixed-weight ESN controller.

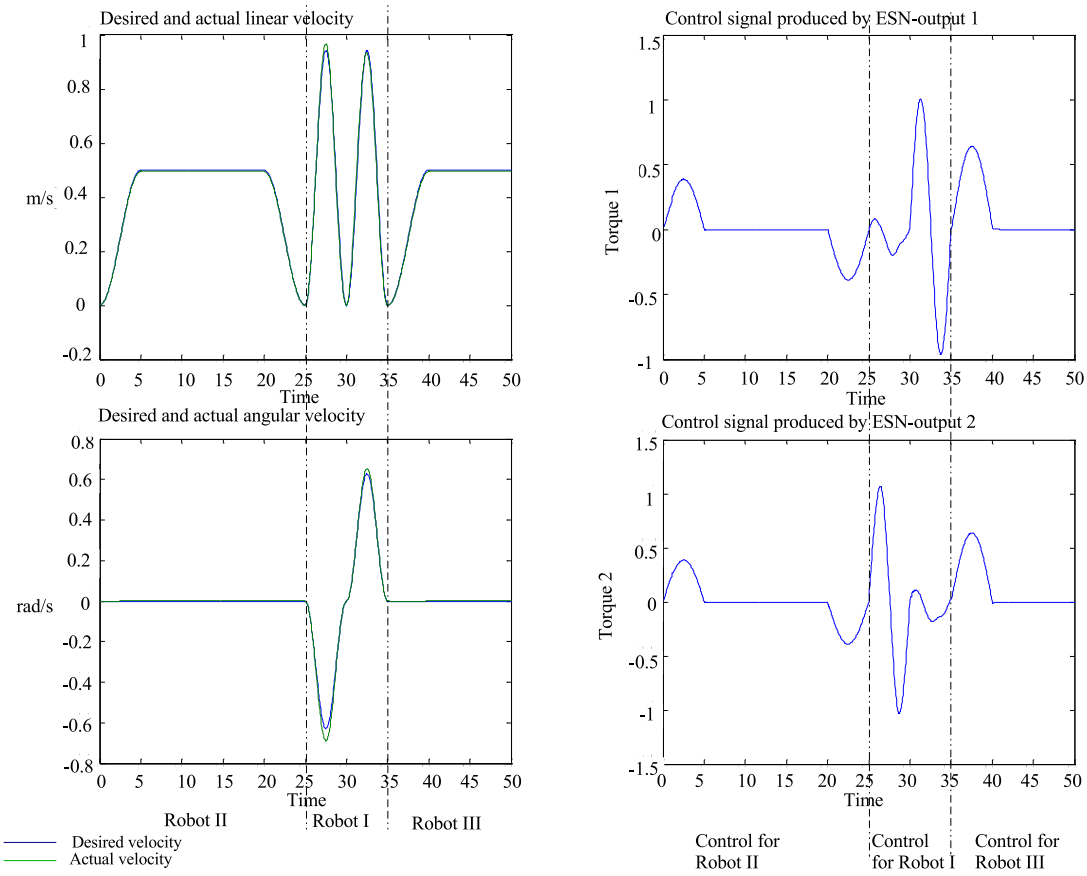


Figure 5.11: Linear and Angular velocity tracking (left) and controls (right). The fixed-weight ESN controls the three robots, following swithes between them. The first switch occurs at time $25s$ from Robot II to I. The second switch occurs at time $35s$ from Robot I to III.

5.7 Kinematic and Dynamic Adaptive Control with ESNs

In this section, an adaptive neurocontrol system with two levels is proposed for the motion control of a nonholonomic mobile robot [87]. In the first level, a position controller is designed to improve the robustness of a feedback kinematic controller and generate linear and angular velocities, necessary to track a reference trajectory. In the second level, another network controller is designed for the model (5.10), which converts the desired velocities, provided by the first level, into a torque control, in presence of time varying parameters (d, m_c). The two control levels are then assembled to form the global navigation control loop.

As in previous section, the advantage of the control approach is that, no knowledge about the dynamic model is required, and no synaptic weight changing is needed in presence of robot's parameters variation. Simulation results are demonstrated to validate the robustness of the proposed approach.

5.7.1 ESN Kinematic Controller

The ESN kinematic controller (called here ESN_K) is designed to emulate the behavior of a feedback trajectory controller, and to improve its robustness in presence of noisy data.
The trajectory controller used is described as following. Let there be prescribed a reference robot (trajectory):

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\phi}_r \end{bmatrix} = \begin{bmatrix} \cos(\phi_r) & 0 \\ \sin(\phi_r) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_r \\ w_r \end{bmatrix} \tag{5.17}$$

where x_r, y_r, and ϕ_r are the pose of the reference robot, and (v_r, w_r) are its linear and angular velocities. We define e_1, e_2, e_3 as following:

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r - x \\ y_r - y \\ \phi_r - \phi \end{bmatrix} \tag{5.18}$$

The inputs (v, w) which make e_1, e_2, e_3 converge to zero are given by [33][29]:

$$\begin{cases} v_d = v_r \cos e_3 + K_1 e_1 \\ w_d = w_r + v_r K_2 e_2 + K_3 \sin e_3 \end{cases} \tag{5.19}$$

where K_1, K_2, K_3 are positive constants.

Training

Figure(5.12) depicts a bloc diagram of the training procedure. First, a random reference trajectories $P_r = [x_r, y_r, \phi_r]$ was generated. The kinematic model of the robot is then controlled by (5.19) in order to minimize the error between the actual pose of the robot $P = [x, y, \phi]$ and the reference trajectory. At the same time, ESN_K learns the behavior of (5.19). To provide ESN_K robustness against disturbances, a gaussian noise with zero mean and a variation level of $\sigma = 0.5$ was added to training data.
Training procedure was performed using an ESN with 5 inputs (e_1, e_2, e_3,v_r, w_r), 9 internal neurons and 2 outputs (desired linear and angular velocities (v_d, w_d)). No back-connection from the output to the DR, and no connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-1, +1]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eigenvalue $|\lambda_{max}| \approx 0.1$.

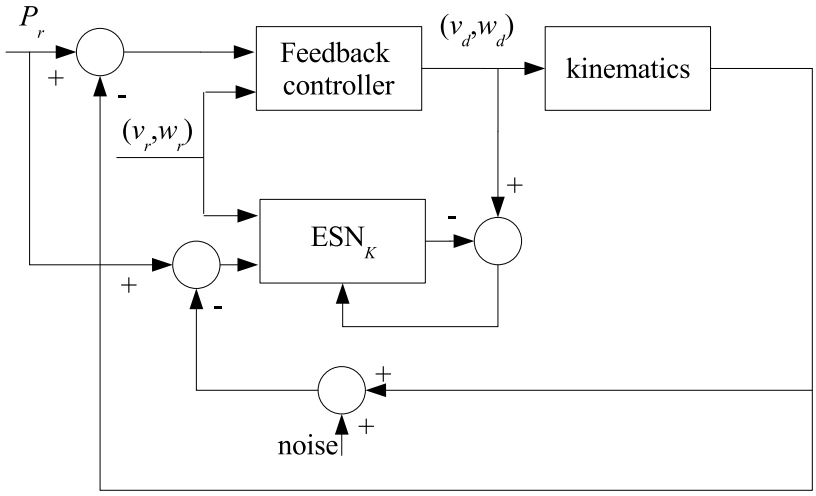


Figure 5.12: Training ESN_K as a kinematic controller.

Test

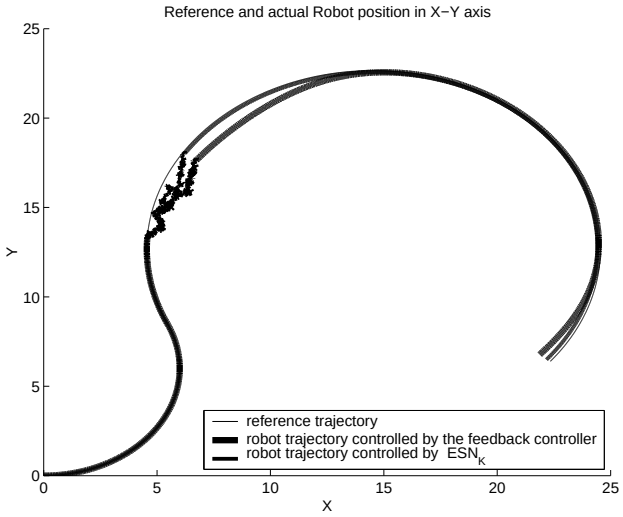
After training, performances of ESN_K and the feedback controller (5.19) are compared in tracking a reference trajectory (see figure 5.13). During the time interval $[0, 1000]$, the two controllers behave similarly. This result is expected, since ESN_k is trained from (5.19). During time interval $[1000, 1200]$ a gaussian noise with zero mean and a variation level of $\sigma = 0.5$ is added to the robot pose values. ESN_k showed better performance to recover the perturbation, compared with the feedback controller (see Fig.5.13.(b) and (c)). In other tests (not presented here), when the variation level of noise is more than $\sigma = 0.5$, the controller (5.19) showed poor performances and lost stability, whereas ESN_k showed more robustness, and could maintain stability of the closed loop.
Here we presented the case, where no initial errors exist. However, when the trajectory tracking starts with initial pose errors, ESN_K could not bring the robot to the reference trajectory, and lost stability. This problem can be solved by using a path planner, which gives a feasible path from the initial robot pose to the nearest point on the reference trajectory.

5.7.2 ESN Dynamic Adaptive Controller

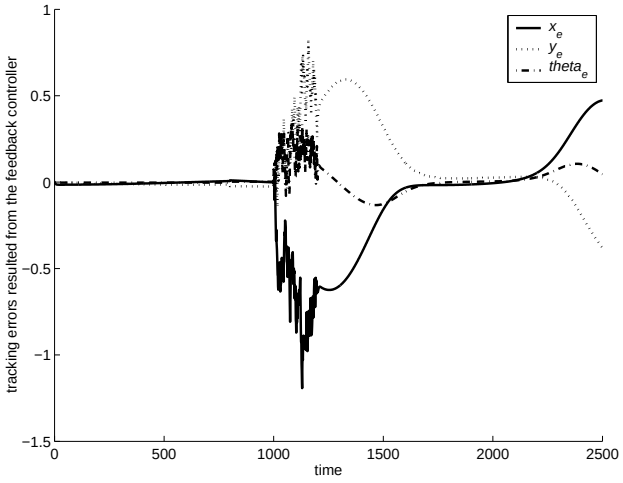
Here, the ESN dynamic controller (ESN_D) is the same controller developed in section (5.4). As mentioned above, metalearning can offer an adaptive behavior with fixed weights. The ESN_D is trained for "all" possible operating conditions of the dynamical model, corresponding to possible combinations of the parameters values (d, m_c).

5.7.3 Kinematic-Dynamic closed loop control

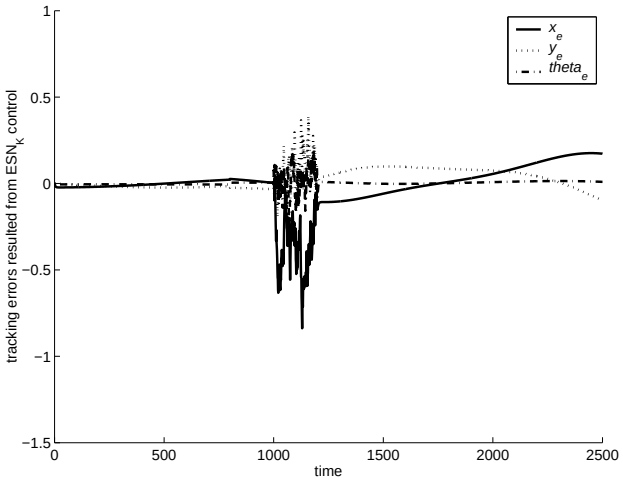
As seen in the two previous sections, the kinematic and dynamic controllers (ESN_K, ESN_D) are designed and trained separately. They are now ready to cooperate together, in order to track a predefined reference trajectory. Thus, the global control structure can be decomposed in two stages (see figure 5.14). An outer-loop, where the ESN_K computes the desired velocity necessary to track a reference trajectory P_r, and an inner-loop, depending on the robot dynamics, where the ESN_D converts the velocity control provided by the outer-loop into a torque control.



(a)



(b)



(c)

Figure 5.13: Kinematic control. a) Robot trajectory tracking controlled by the two controllers separately. b)Tracking errors resulted from the feedback controller ($K_1 = K_2 = K_3 = 5$). c) Tracking errors resulted from the ESN_K control.

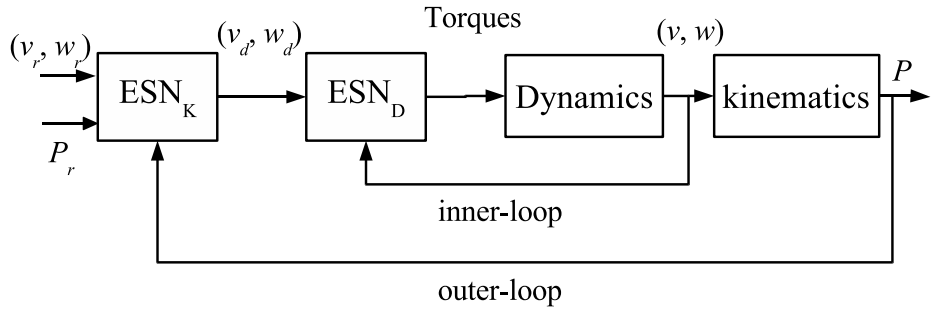


Figure 5.14: Global control structure of a nonholonomic mobile robot using ESNs

5.7.4 Results

In the simulation, the parameters m_c and d are time varying and chosen according to Table 5.3. The reference velocities v_r and w_r are chosen as following:

$$0 \leq t \leq 100 : v_r = 0.25(1 - \cos \tfrac{\pi t}{5})$$
$$w_r = -0.5 v_r$$

$$100 \leq t \leq 200 : v_r = 0.5$$
$$w_r = -0.25$$

$$200 \leq t \leq 300 : v_r = 0.5$$
$$w_r = -0.25 \cos \tfrac{\pi t}{5}$$

$$300 \leq t \leq 700 : v_r = 0.5$$
$$w_r = 0.25$$

$$700 \leq t \leq 800 : v_r = 0.125(1 - \cos \tfrac{\pi t}{5}) + 0.25$$
$$w_r = -0.25 \cos \tfrac{\pi t}{5}$$

$$800 \leq t \leq 1000 : v_r = 0.25$$
$$w_r = -0.25$$

Table 5.3:

Time	m_c	d
$0 \leq t \leq 400$	30	0.3
$400 \leq t \leq 500$	34.4	0.13
$500 \leq t \leq 600$	30	0.3
$600 \leq t \leq 700$	33.2	0.41
$700 \leq t \leq 1000$	30	0.3

Figure (5.15) shows the control results. ESN_K tracks the reference trajectory by providing desired linear and angular velocities. Because initial errors are zero, desired velcities are almost similar to the reference velocities. On the other hand, ESN_D shows an excellent tracking of the desired velocity provided by ESN_K. Furthermore, when m_c and d values are suddenly changed, ESN_D locks on the new dynamic behavior of the robot and provides the appropriate control signal (Figure 5.15 (c) and (d)), without changing any synaptic weight. We can see small adaptation periods after each variation. These are necessary for the network to switch from one familly of orbits to another. Surprisingly, the global control loop is barely affected by these switches, and the robot tracks reasonably the desired trajectory (figure 5.15 (e)).

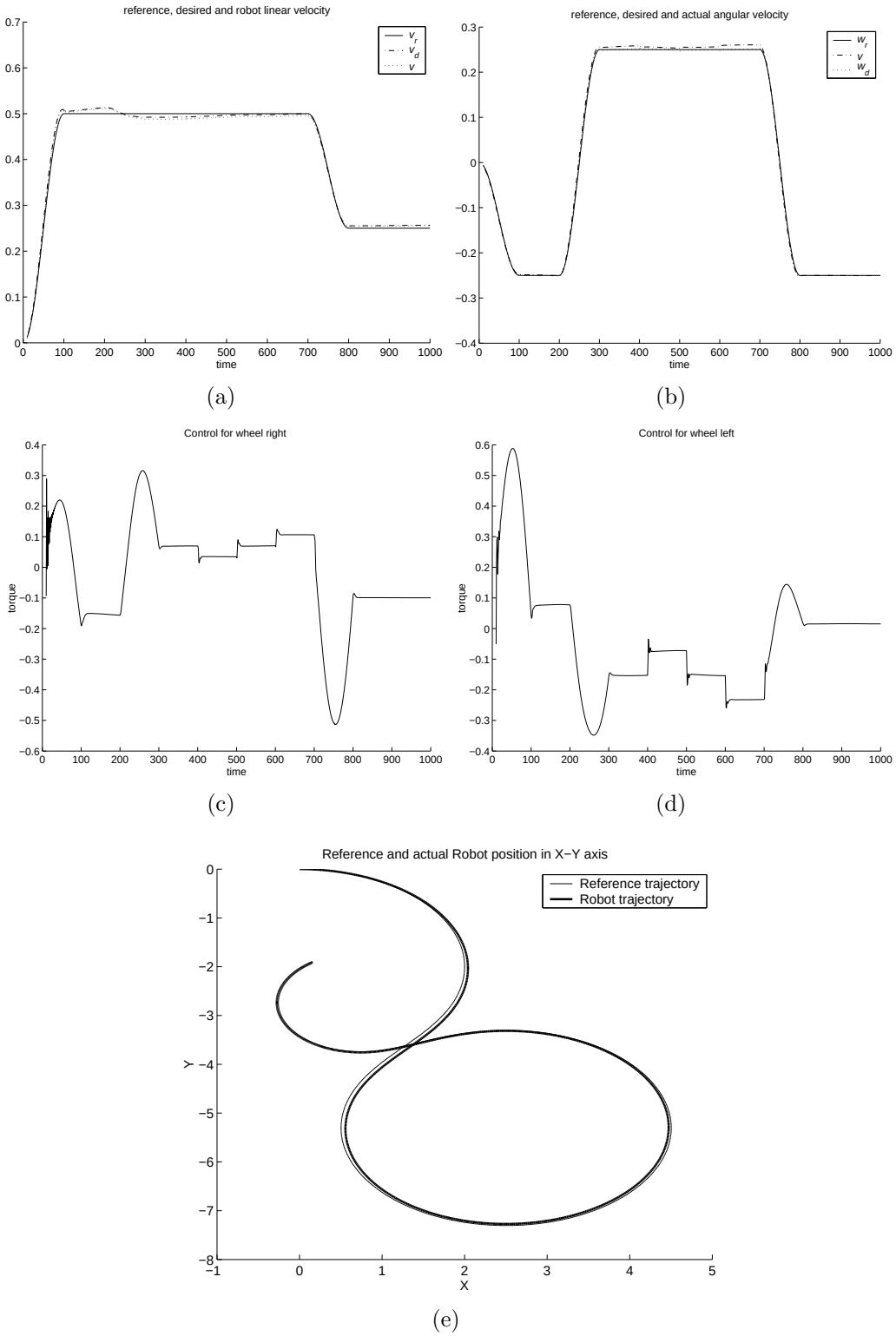


Figure 5.15: Kinematic and dynamic control. a) Linear velocity tracking. b) Angular velocity tracking. c) Computed torque for wheel right. d) Computed torque for wheel left. e)Trajectory Tracking

5.8 Discussion

In this chapter an adaptive neurocontrol system based on ESNs is developped. A kinematic level controller (ESN_K) is trained to mimic a feedback controller, and to improve its robustness against noise. Indeed, the trained ESN_K showed more robustness against external perturbations. However, excelent control performances are obtained only when initial errors are close to zero. When initial errors are far from zero, poor performances were obtained. This problem can be solved by using a path planner, which gives a feasible trajectory to the nearest point on the reference trajectory. Many approaches are introduced in [28]. To reach the assumption "perfect velocity tracking", the robot dynamics is taken into consideration by a dynamic level controller (ESN_D). The role of ESN_D is to convert the desired velocities, provided by the first level, into a torque control. At this level, no knowledge about the dynamic model was required, since the controller is designed only by learning I/O data collected from the robot. This property is very important in practical cases, where it is almost impossible to have the real parameters values of a robot. Because of the "rich" variety of its internal dynamics, the trained network could make an excellent generalization on incoming data and deliver the appropriate control signals, in order to track the reference velocity. Moreover, the ESN_D controller was asked to exhibit a characteristic, normally ascribed to adaptive controllers, whose parameters change in response to an environmental change. "Adaptation" in this work is defined as the ability of the controller to recognize change only through the robot output and its own state, without changing any synaptic weight. This capability is a natural consequence of prior 'metalearning' used during training.
For multi-robot control, the fixed weight ESN controller showed a reasonable balance between the variety of the reference velocity and the variety of the robots. When a "new" input (from one robot, which is already learned) is provided, the state of the network switch from one family of orbits to another, which corresponds to the new input. We recognize here that the controller trained for many robots may not be as effective on a given robot as a controller trained only for that robot.

However, in all these developments it was not easy to find the optimum parameters of the ESN in order to obtain a "rich" variety of internal dynamics. Using a "relatively" large dimension (more than 30 internal neurons) the

network lost stability at many times and exhibited sometimes high-frequency oscillations on smooth test signals. With small dimension (say 4-6 internal neurons), the network could not react quickly to the velocity variations. Moreover, some weight initializations result to instabilities at many times, especially when the switch between Robots I, II and III occurred.

5.9 Conclusion

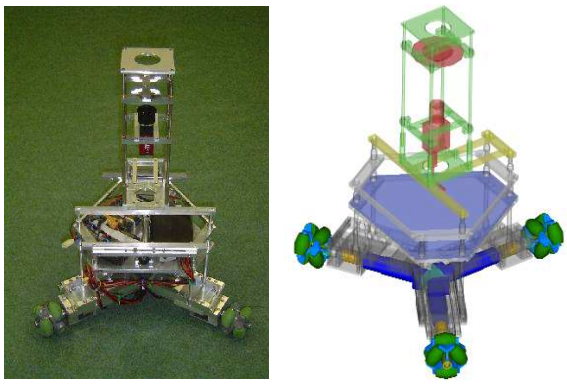
This chapter has presented the motion control of nonholonomic mobile robots using echo state networks. The approach proposed includes also the dynamic-level control, without knowledge about the dynamic model of the robot. This property makes it very useful in practical situation, where the exact knowledge about the real parameters is almost unattainable. Furthermore, the ESN dynamic controller was asked to exhibit a characteristic, normally ascribed to adaptive controllers, whose parameters change in response to an environmental change. "Adaptation" in this work is defined as the ability of the controller to recognize change only through inputs and its own state, without changing any synaptic weight. This capability is a natural consequence of prior 'metalearning' used during training. This strategy was also used to build up a single fixed weight ESN to act as a dynamic controller for three distinct robots. After training, the fixed weight ESN controller showed a reasonable balance between the variety of the reference velocity and the variety of the robots. Finally, an adaptive neurocontrol system with two levels based on ESNs was developed. First, a kinematic level controller (ESN_K) is trained to mimic a feedback controller, and to improve its robustness against noise. The dynamic controller (ESN_D) could make an excellent generalization on incoming data and deliver the appropriate control signals (torques), in order to track the desired velocity. Moreover, ESN_D recognizes parameters variation only through the robot output (velocity) and its own state, without changing any synaptic weight.
All in all, training experiments carried out here demonstrate that two control levels based on small and partially interconnected ESNs can be designed to act as a motion control system for mobile robots, in presence of external perturbations and time varying parameters. However, we are aware of a certain degree of arbitrariness in the choice of the controller network parameter and architecture. Substantial investigation on ESNs and more experiences are still needed to ensure that the results we have achieved to date are statis-

tically significant. Next chapter will present a first real implementation of ESNs: Control of a real Omnidirectional Mobile Robot.

Chapter 6

Control of an Omnidirectional Robot with ESNs

In this chapter, the ESN strategy is proposed for the adaptive velocity control of an omnidirectional mobile robot available at the Robotics Lab of the University of Stuttgart (figure 6.1). As seen in chapter (5), no knowledge about the dynamical model is required, since the controller is designed by learning only input-output data collected from the robot. Furthermore, the trained ESN controller has the ability to recognize parameters variation (here the mass and the center of mass) only through its inputs, and to adjust its behavior to these changes, without changing any synaptic weight. Some experimental results are presented.



(a) (b)

Figure 6.1: Omnidirectional robot. a)hardware photo. b) CAD model

6.1 Introduction

Compared with the nonholonomic mobile robots, omnidirectional mobile robots provide superior manoeuvring capability. The ability to move simultaneously and independently in translation and rotation makes them widely studied in dynamic environmental applications. The annual RoboCup competition (www.robocup.org) is an example where omnidirectional robots are widely used. However, quite few research studies on this type of robots have been reported. Most of them have been focused on the mechanical design and on the kinematic level control, assuming that there is "perfect" velocity tracking. However, as it is well known, the control at the kinematic level may be instable if there is errors control at the dynamic level. Therefore, the velocity control is at least as important as the position control. Recently, dynamic modeling and some analysis for omnidirectional robots have been addressed in [88, 89, 90]. In contrast with these theoretical developments, only few experimental works have been presented.
In practical situations, however, exact knowledge about the robot parameters values is almost unattainable. If we assume that some of these parameters are time varying, the problem becomes more complicated. To deal with this problem, there are possible methods, which can be used, even when the knowledge about the dynamic model is not complete, like robust adaptive control [30]. Another possible approach is to consider the robot as a "black box", in order to avoid the estimation of its real parameters.

In this chapter, the ESN velocity controller is designed only by learning I/O data collected from the robot. Furthermore, metalearning offers the ESN the ability to adapt to time varying parameters without changing any synaptic weight.

The rest of this chapter is organized as follows. Section (6.2) presents the omnidirectional robot and the problem to solve. The control approach with some experimental results are presented in Sections (6.3) and (6.4). Finally, results are discussed in section (6.5), and section (6.6) summarizes the conclusions of the present chapter.

6.2 Omnidirectional Robot

A holonomic system is one in which the number of degrees of freedom are equal to the number of coordinates needed to specify the configuration of the

system. In the field of wheeled mobile robots, the term holonomic mobile robot (also called omnidirectional mobile robot) is used for robots having three degrees of freedom. In contrast to nonholonomic robots, a holonomic robot can move in an arbitrary direction continually without changing the direction of the wheels. It can move back and forth, slide sideways, and rotate in place. They are desirable because they do not have kinematic motion constraints, which make path planning and motion control much simpler. To achieve a holonomic motion only wheels with three degrees of freedom must be employed. Many different mechanisms have been created; most of them are based on the same general principle: a wheel provides traction in the direction normal to the motor axis, and it rolls passively in the diretcion of the motor axis. Examples of such wheeles are Swedish wheels [91][92][93], Orthogonal-wheels [94], Mecanum wheels [95], and Ball wheels [96][97]. However, all these passive mechanisms, except for some types with ball wheels, have discontinuous wheel contact points, which are a great source of vibration. This problem becomes significant when the robot moves at high speeds (see results later).

6.2.1 Hardware

Experimentations are performed on one omnidirectional robot (Figure 6.1) of Soccer-robots team available at the robotics Lab of University of Stuttgart. The robot is equipped with 3 omni-wheels equally spaced at 120 degrees from one to another, and driven by three $90W$ DC motors. A personal computer on board is used to manage different sensors and tasks. For environment sensing, the robot is equipped by an omnidirectional vision system, based on a hyperbolic mirror and a standard IEEE1394 (FireWire) camera (Figure 6.2). Omnidirectional vision provides our robot a very large field of view, which has some useful properties. For instance, it can facilitate the tracking of robots, the ball, and a set of environmental features used for self-localization. More hardware specifications of the robot are listed in Table 6.1.

Max Acceleration	$3.5m/s^2$
Max. Speed	$2.5m/s$
Wheels	Omni-wheels (Swedish)
Host Computer	2,6GHz , RAM 600MB,60 GB HD
Power supply	Two 13 V batteries in series
Steering control	Interfaces with 3 DC motors
Dimensions	48,5cm wide, 80cm height
Weight	15 kg

Table 6.1: Robot Specifications

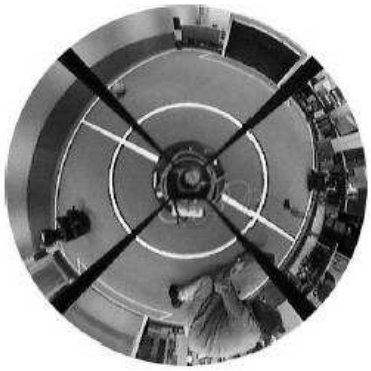


Figure 6.2: An image from the omnidirectional camera.

6.2.2 Kinematic Model

The geometry of the omnidirectional robot of figure (6.1) and its coordinate definitions are shown in figure (6.3). Let $\dot{\theta}$ denotes the angular velocity of the internal reference frame of the robot (X, Y) relative to an absolute refernce frame (x, y). The linear velocity is denoted by v and its direction with respect to the platform internal reference frame is denoted by $\varphi \in [0, 2\pi]$. The angular velocities of the wheels can be calculated as [94]:

$$w_1 = \frac{v}{2R}(\sin\phi - \sqrt{3}\cos\phi) + \frac{\dot{\theta}L}{R} \tag{6.1}$$

$$w_2 = -\frac{v}{R}\sin\phi + \frac{\dot{\theta}L}{R} \tag{6.2}$$

$$w_3 = \frac{v}{2R}(\sin\phi + \sqrt{3}\cos\phi) + \frac{\dot{\theta}L}{R} \tag{6.3}$$

where R is the radius of the spherical wheels and L represents the distance between the center of the platform and the center of the wheel i. w_i is the angular speed of wheel i, and x ,y and θ are the pose of the centre of mass (CM) of the robot. In the case of our robot, $R = 0,04$ and $L = 0,2$.

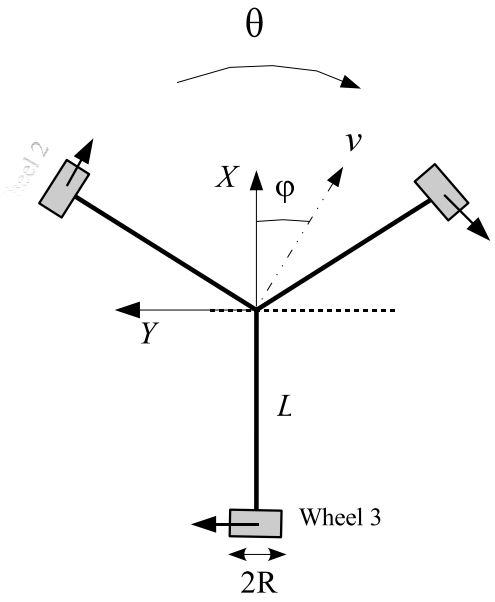


Figure 6.3: Kinematic geometry of the Omnidirectional Robot.

If we set $\dot{x} = v\cos\phi$ and $\dot{y} = v\sin\phi$, then equations (6.1)(6.2)(6.3) can be written as:

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \frac{1}{R}\begin{pmatrix} \frac{-\sqrt{3}}{2} & \frac{1}{2} & L \\ 0 & -1 & L \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & L \end{pmatrix}\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} \tag{6.4}$$

From the relation (6.4) we can see that the rotational and translational motions are fully decoupled and can be controlled independently and simultaneously.

6.2.3 Control System

The control system is decomposed in two stages. An inner loop, depending on the robot dynamics and used to control linear and angular velocities, and an outer loop, which guarantees that the robot follows the desired trajectory. Both controllers are based on PID control strategy (Figure 6.4).

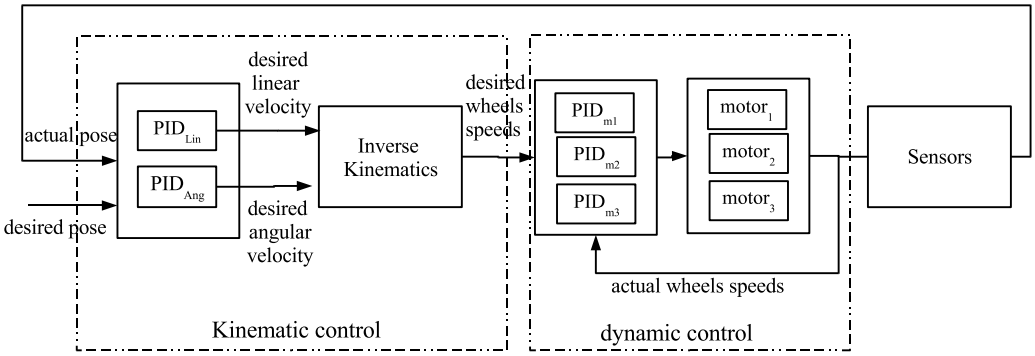


Figure 6.4: Kinematic and dynamic control loops.

6.2.4 Problem Statement

In this chapter, we want to implement our ESN velocity control strategy on the robot of figure (6.1). We are interested also in much more interesting problem of requiring a fixed-weights ESN to make a velocity adaptive control. No knowledge about the dynamics model is available, and the robot has a time-varying mass (a switch between $15Kg$ and $17.5Kg$), which results to variations of the center of mass. Also, speed measurements are noisy due to the nature of the wheels and the sensors. Upon completion of the training procedure, we expect that the ESN controller will be capable to exhibit characteristics, normally ascribed to adaptive controllers: i) make an acceptable generalization, in order to track reasonably a reference velocity; ii) detect the mass variation; and ii) adapt itself to the perceived change and generate the appropriate control signal. Naturally, we expect adequate performance only for the two possible global mass of the robot for which the network has been trained.

6.3 Velocity Control with ESN

In practical situations, the assumption *"perfect"* velocity tracking at the dynamic-level is almost unattainable by the PID controllers implemented on board. The ability of a PID controller to cope with some complex properties of the robot such as non-linearities, friction, and time-varying parameters is known to be very poor. To improve the velocity control, we propose to use an Echo State Network to deal with the whole dynamics of the robot as a "black box". Upon completion of the training procedure, we expect that the ESN controller will be capable to minimize reasonably errors between the desired and the actual robots velocities, without knowledge about the robot parameters.

6.3.1 Training

I/O (PWMs/wheels speeds) training data are collected by moving the robot "arbitrarily" with different velocities in different directions. 1000 sequences were collected from the robot and stored in a file. To train the ESN as a velocity controller, we used the same bloc diagram depicted in figure (6.5). The ESN learned the teacher signals(PWMs), which bring each wheel from an actual speed at time (n) to a future speed at time $(n+1)$.

The ESN architecture was chosen as follows. 6 inputs (actual and delayed wheels speeds), 13 internal neurons and 3 outputs (PWMs duty ratio for each motor). No back-connection from the output to the DR, and no connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-1, +1]$. The internal weight matrix W has a sparse connectivity of 20% and scaled such that its maximum eingenvalue $|\lambda_{max}| \approx 0.3$.

6.3.2 Control Procedure

After training procedure has been completed, the network was implemented in the control system on the host computer on-board (Figure 6.6). Desired speeds of the wheels were computed from the equation (6.4). Using the actual (measured) and desired wheel-speeds, the ESN send the appropriate PWMs duty via a serial (RS232) connection to an electronic interface, which produces the correspondent amplified PWMs voltage to the three motors.

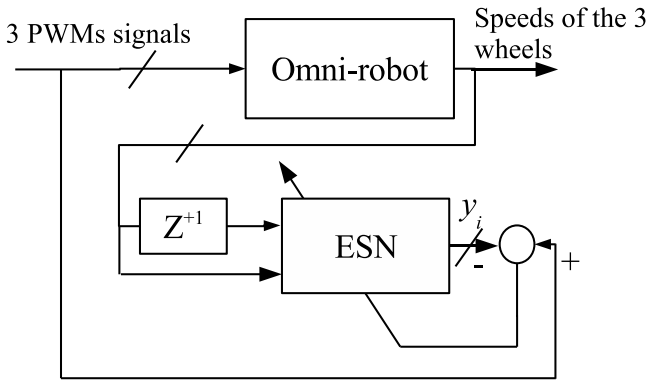


Figure 6.5: Structure of the control system.

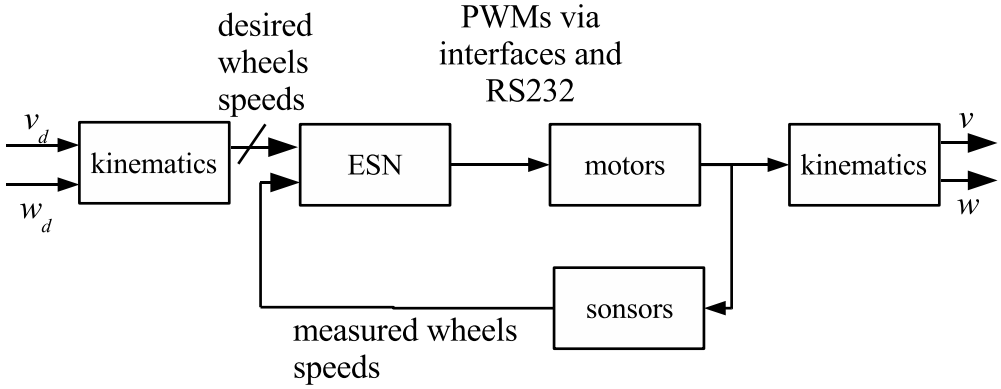


Figure 6.6: Structure of the control system.

6.3.3 Results

After training, several experimental tests are performed; two of them are reported here [98][85]. They cover some typical movements of a RoboCup player during a soccer game.

Experiment 1

In this experiment, the objective is to track the constant reference velocity:

$$0s \leq t \leq 10s : v_d = 0.22\ m/s, \quad (\varphi = 0)$$
$$w_d = 0\ rad/s$$

In this experiment, the robot should move straight a head(no angular velocity). Therefore, wheel 3 should be blocked. During this experiment (Figure 6.7), the ESN controller could bring the robot to the desired velocity, even the presence of high friction between the carpet and wheel 3, which explains the high frequency in the control signal delivered by the ESN for this wheel.

Experiment 2

In this experiment, the objective is to track the time varying reference velocity:

$$
\begin{aligned}
0s \leq t \leq 7s : v_d &= 0.22\ m/s, \quad (\varphi = 0) \\
w_d &= 0\ rad/s \\
7s \leq t \leq 15s : v_d &= 0.22\ m/s, \quad (\varphi = \pi) \\
w_d &= 0\ rad/s \\
15s \leq t \leq 23s : v_d &= 0\ m/s \\
w_d &= -2\ rad/s \\
23s \leq t \leq 30s : v_d &= 0.25\ m/s, \quad (\varphi = 0) \\
w_d &= -3\ rad/s
\end{aligned}
$$

As shown in Figure (6.8), during $[0s, 7s]$, the robot behaved like in experiment 1. In $[8s, 15s]$, the same desired linear velocity is given to the controller, but in other direction (according to the angle φ in Figure 6.3). At time $8s$ the controller stopped the robot, and tried to reach again the same translation velocity in the new direction ($\varphi = \pi$). During $[15s, 23s]$, the robot had to be turned around it self, since the desired translation velocity is zero, and the desired rotation velocity is $-2\ rad/s$. In this situation, the wheels are more sensitive to the carpet, and exhibit higher frequency around the reference. In the last time interval $[23s, 30s]$, the ESN had to control the robot in a curve. During training, the ESN did not learn this situation. Despite this lack of information, the ESN could successfully control the robot independently in translation and rotation.

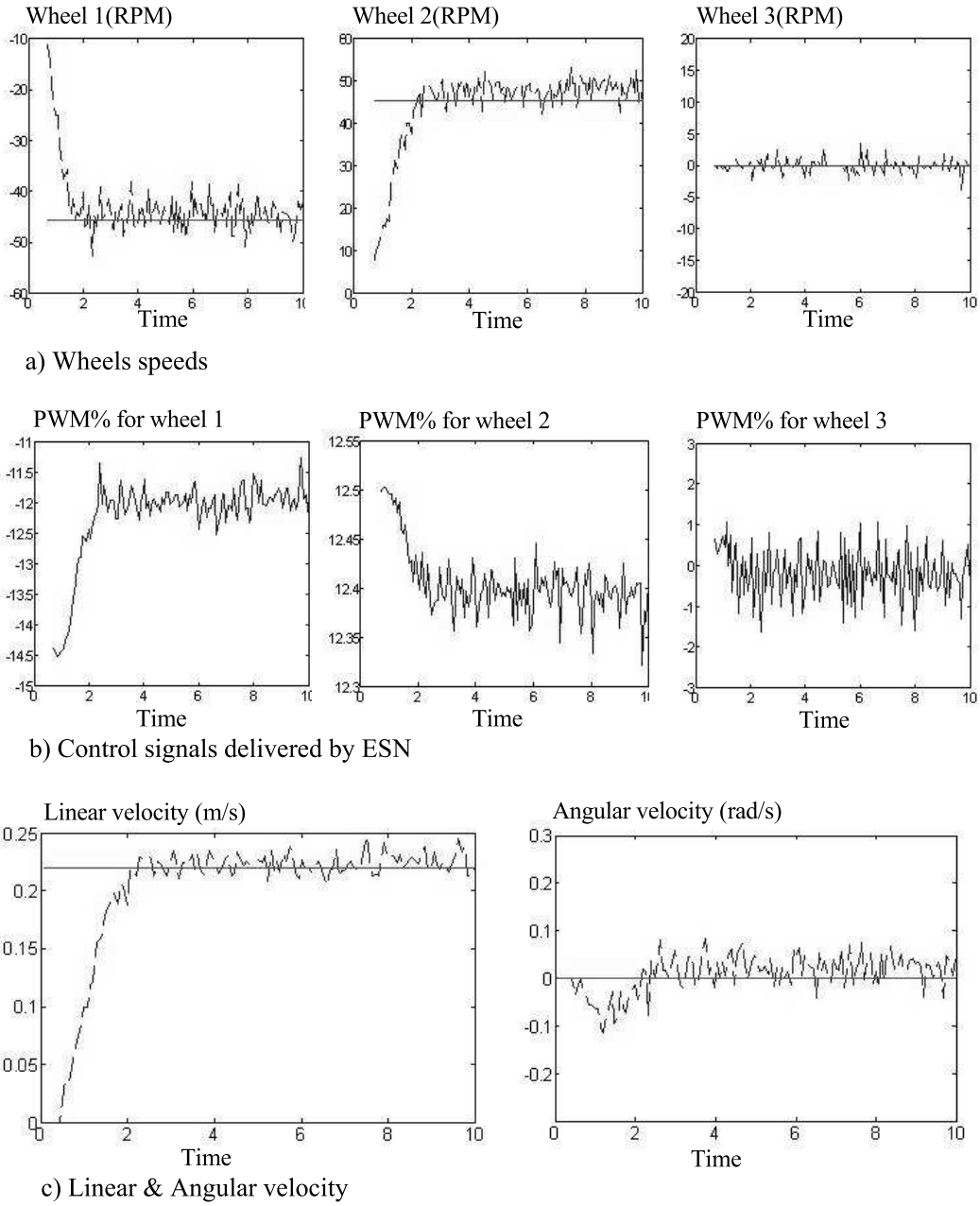


Figure 6.7: Results of Experiment 1. Desired velocity(solid) and actual robot velocity (dashed).

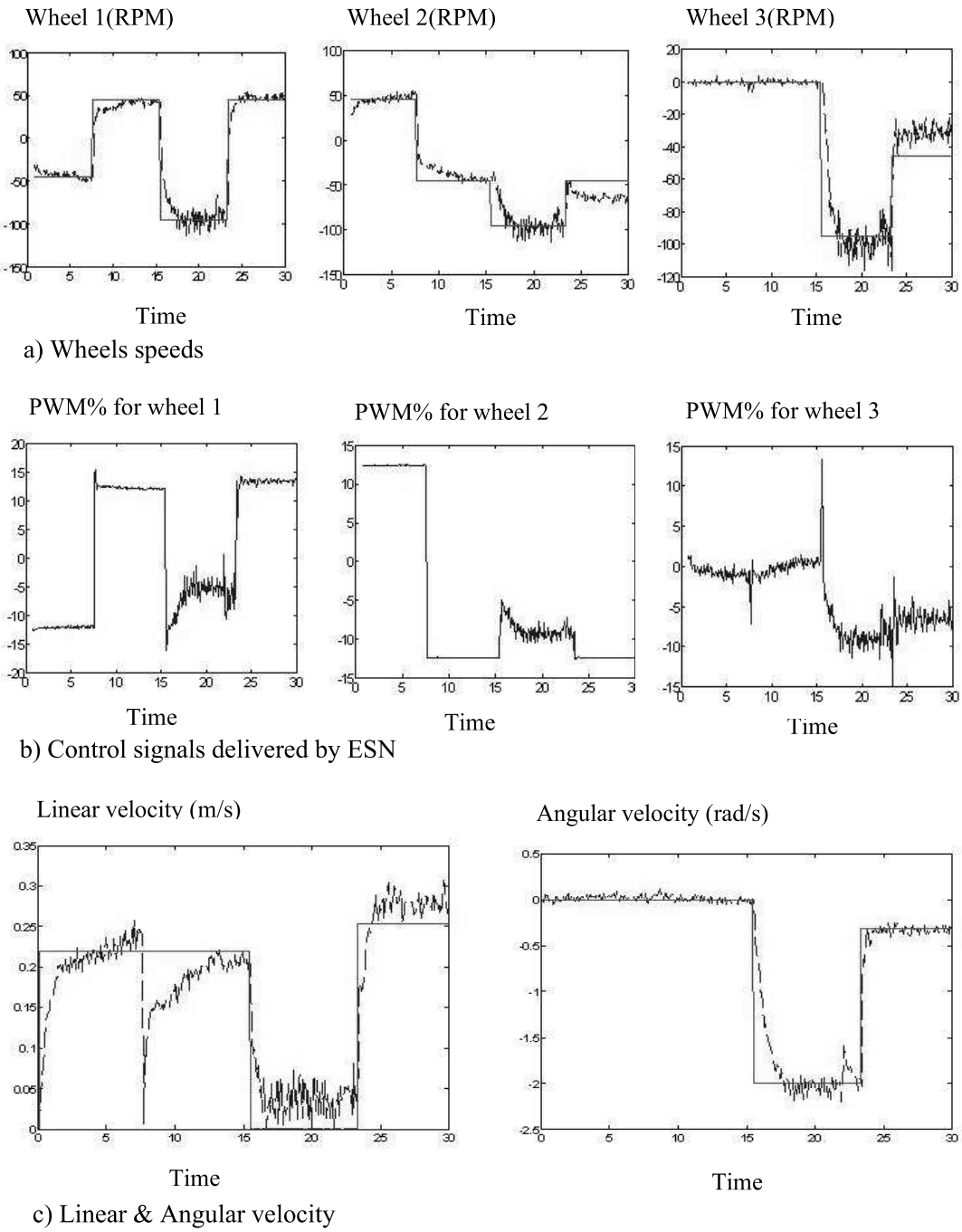


Figure 6.8: Results of Experiment 2. Desired velocity(solid) and actual robot velocity (dashed).

6.4 Fixed-Weight ESN Adaptive Controller

In section (6.3), an ESN was trained to act as a velocity controller for the omnidirectional robot. The advantage of the control approach used is that no knowledge about the dynamic model of the robot is required. In this section, however, we are interested in much more interesting problem of requiring a fixed-weights ESN to make an adaptive dynamic control for the robot, in presence of time-varying mass (a switch between $15Kg$ and $17.5Kg$). This change can result to variations of the center of mass (CM). Upon completion of the training procedure, we expect that the ESN controller will be capable to exhibit characteristics, normally ascribed to adaptive controllers: i) detect the mass and CM variation; ii) adapt itself to the perceived change and generate the appropriate control signal; and iii) make an acceptable generalization, in order to track reasonably a reference velocity. Naturally, we expect adequate performance only for the two possible global mass ($15Kg$ and $17.5Kg$) of the robot for which the network has been trained.

6.4.1 Procedure

In this work, we propose to use the metalearning procedure to develop an adaptive velocity ESN controller. "Adaptation" here is the ability of the resulting ESN to recognize the mass variation only through its inputs, and its own state, without changing any synaptic weight. Again, no multi-stream training is needed, since the ESN batch update uses all data at once, and does not suffer from the recency effect.
Training data were prepared by driving the robot with different velocities in different directions. 1000 I/O sequences were collected from the robot having its initial mass ($15Kg$) and other 1000 I/O sequences when the extra mass $m = 2.5Kg$ was added on the robot. As a result, 2000 I/O heterogenous data are used for training. To train the ESN as a velocity controller, we used the training approach presented in figure (6.5). The ESN learned the output teacher signals (PWMs), which bring each wheel from an actual speed at time (n) to a future speed at time $(n+1)$. Here the ESN architecture was chosen as follows. 6 inputs (actual and delayed wheels speeds), 13 internal neurons and 3 outputs (PWMs duty ratio for each motor). No back-connection from the output to the DR, and no connections from the input directly to the output. The input and the internal synaptic connections weights were randomly initialized from a uniform distribution over $[-1, +1]$. The internal weight ma-

trix W has a sparse connectivity of 20% and scaled such that its maximum eingenvalue $|\lambda_{max}| \approx 0.3$. After training procedure has been completed, the network was implemented in the control system on the host computer onboard (Figure 6.6). Desired speeds of the wheels are first computed from the desired (reference) linear and angular velocities (v_d, w_d), using the equation (6.4). Using the actual (measured) and desired wheel-speeds, the ESN send the appropriate PWMs duty via a serial (RS232) connection to an electronic interface, which produces the correspondent amplified PWMs voltage to the three motors [99].

6.4.2 Results

After training, several experimental tests were performed. We report here two of them. In the first, the fixed-weight ESN controller was used to control the robot having the two different mass separately. In the second, the adaptivity of the ESN is tested during the movement of the robot. Control results for the first case are present in figure (6.9) and those for the second case in figure (6.10). In each figure, the three first panels show the wheels-speeds, while the other panels provide the corresponding control signals (PWMs %) produced by the ESN controller.

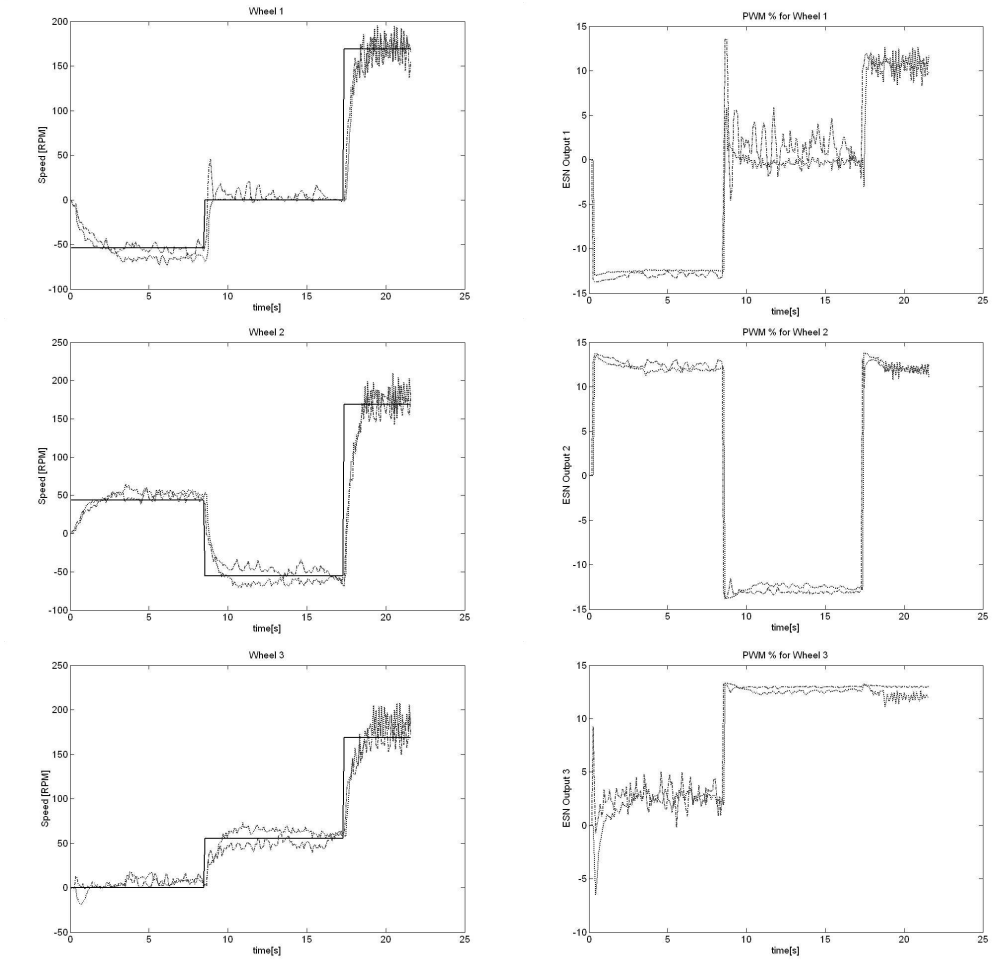
Experiment 1

In this experiment (figure 6.9), the fixed-weight ESN is tested separately on two operating conditions of the robot ($15Kg$ and $17Kg$). In both tests, desired linear and angular velocities (v_d, w_d) were chosen as follows:

$$
\begin{aligned}
1s \leq t \leq 7s &: v_d = 0.23\ m/s, \quad (\varphi = 0) \\
&\quad w_d = 0 \\
7s \leq t \leq 14s &: v_d = 0.27\ m/s, \quad (\varphi = \tfrac{2\pi}{3}) \\
&\quad w_d = 0 \\
14s \leq t \leq 22s &: v_d = 0\ m/s, \quad (\varphi = 0) \\
&\quad w_d = 3.5
\end{aligned}
$$

In the first operating condition ($15Kg$), the ESN controller has made a good generalization and could bring the robot to the desired velocity, even the presence of high friction between the carpet and wheel 3. In the second operating condition ($17.5Kg$), the ESN recognized this change only through the

wheels speeds and its own state, and could provide the appropriate control signals, in order to bring the robot to the desired velocity.



(a) (b)

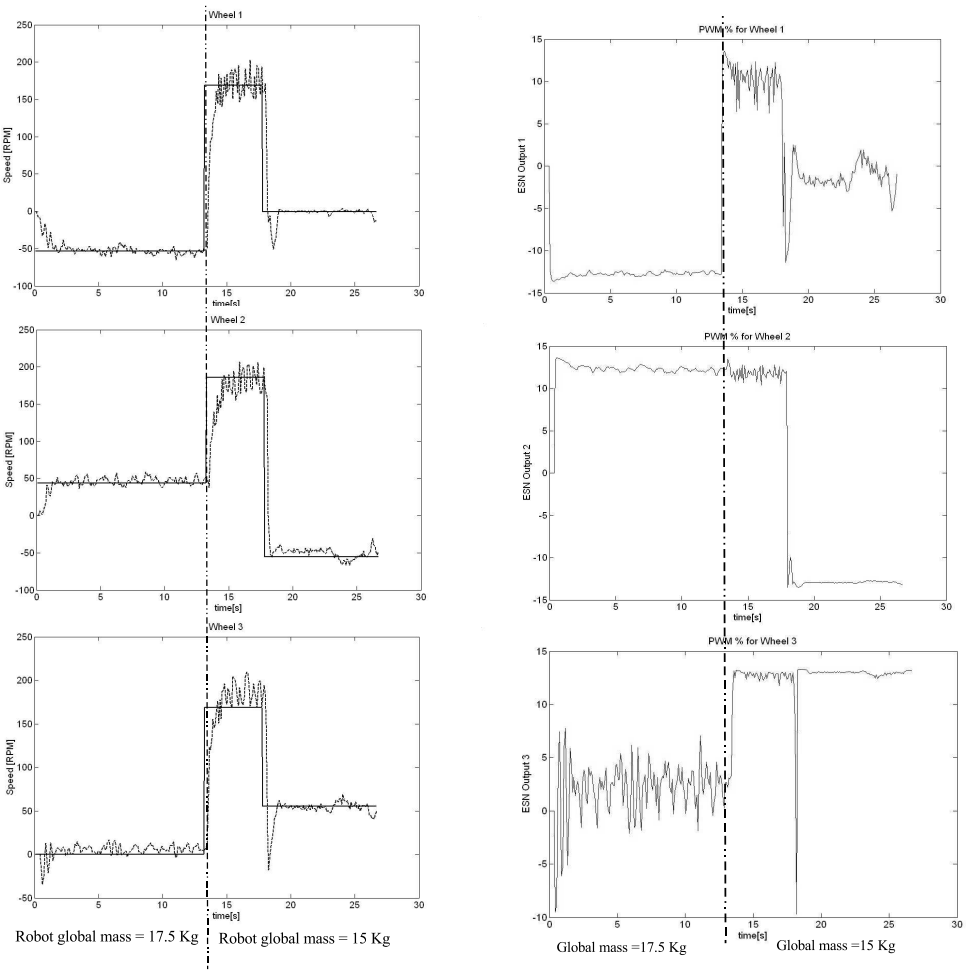
Figure 6.9: Control results of the case 1. a) Desired speeds(solid), Wheels speeds (initial mass)(dashed), and Wheels speeds (mass=17.5 Kg) (dash dot). b) ESN control signals initial mass)(dashed). ESN control signals (mass=17.5 Kg) (dash dot)

Experiment 2

In this experiment (figure 6.10), the performance of the fixed-weight ESN to handle variation of the mass is tested during the movement of the robot. (v_d, w_d) were chosen as follows:

$$
\begin{aligned}
1s \leq t \leq 13s : v_d &= 0.23\ m/s, \quad (\varphi = 0) \\
w_d &= 0 \\
14s \leq t \leq 18s : v_d &= 0\ m/s, \quad (\varphi = 0) \\
w_d &= 3.5 \\
19s \leq t \leq 26s : v_d &= 0.27\ m/s, \quad (\varphi = \tfrac{2\pi}{3}) \\
w_d &= 0
\end{aligned}
$$

During the first 13 seconds, the ESN was asked to control the robot having a mass of $17.5Kg$. In this interval, the ESN could successfully bring the robot to the desired velocity $(0.23m/s)$. At time $t = 13s$, the mass of the robot is reduced to $15Kg$. This switch required the controller outputs to change, since the robot dynamic characteristics have been changed. Surprisingly, the control is barely affected by this variation, and the controller showed a rapid adaptation to these change, and delivered the appropriate control signals to continue tracking the desired velocity.



(a) (b)

Figure 6.10: Control results of the case 2. a) Desired speeds(solid) and actual Wheels speeds (dashed), b) ESN control signals

6.5 Discussion

During experiments, we had to deal with many practical problems. The major problem was training data. It is technically not possible to use random inputs, in order to collect training data. The only realistic possibility available was to drive the robot with different "low" velocities (max 0.5 m/s), in order to avoid slippage of the wheels, and to keep the robot on a limited space. It is clear that using this method, training data will be not rich enough to give complete information about the robot dynamics. Other problem is the nature of the robot. Omnidirectional characteristics are obtained only by using omni-wheels. However, it is known that these wheels are very sensitive to the road (carpet) condition and their performances are limited, compared with the conventional wheels. This limitation produces a lot of errors in speed measurements. In addition, some errors were also produced by the sensors, since they deal only with integers.
Another problem is the network architecture. During preparation of the network, it was not easy to find its optimum parameters. Using a "relatively" large dimension (more than 30 internal neurons) the network lost stability at many times and exhibited sometimes high-frequency oscillations on smooth accelerations. This is due to the high degree of freedom of the closed loop Robot-ESN. With small dimension (say 5-8 internal neurons), we could minimize these oscillations, but the network could not make good generalization on data used during training. With 13 internal units, the ESN showed an acceptable behavior.

6.6 Conclusion

This chapter has presented a velocity controller for an omnidirectional robot. The ESN control approach requires no prior information about the dynamics of the robot. This property makes it very useful in practical situation, where the exact knowledge about the mobile robot parameters is almost unattainable. Despite the "poor" quality of training data, and the performance limitation of the omni-wheels, the ESN controller could achieve acceptable control results. Beyond this "black-box" modeling capability, the ESN controller is being asked to exhibit an adaptive behavior to control the robot in a presence of a time varying parameter (here the mass). "Adaptation" in this work is defined as the ability of the controller to recognize change only through

the robot wheels-speeds and its own state, without changing any synaptic weight. This capability was acquired through prior metalearning used during training. The fixed-weight ESN controller could achieve acceptable adaptive control results and showed a reasonable balance between the variety of the reference velocity and the robot mass variation. The resulted controller could "instantaneously" adjust itself to the perceived change, only through its inputs, without changing any synaptic weight. Note that the ESN controller is effective only for the two operation conditions of the robot ($15Kg$ and $17.5Kg$), for which it has been trained, not for arbitrarily chosen robot-mass.

All in all, the real implementation carried out here demonstrates that a small and partially interconnected ESN can be trained to act as an adaptive controller for a real system, whose parameters could be time varying. The resulted controller could "instantaneously" adjust it self to the perceived change, only through its inputs, without changing any synaptic weight. Such behavior in 'classical' control requires a more complicated adaptive system.

Chapter 7

Neural Fields for Behavior Generation

7.1 Introduction

The basic task the robot has to perform is to reach a goal under constraints, e.g moving towards a goal while avoiding obstacles. Approaches that have been developed for this problem can be divided into global and local methods. Global methods require the environment to be completely known and the terrain should be static, and then they return a continuous free path. By contrast, local methods need only local information. It means that the path planning is done while the robot is moving, in response to environmental changes. Due to their low-computational costs, local methods are much more suitable for real application where the environmental state changes continually. The most popular local method is the potential field approach proposed by Khatib [20]. The idea is to consider that the robot moves under influences of an artificial potential field. The target applies an attractive force to the robot, while obstacles exert repulsive forces onto the robot. The sum of all forces determines the subsequent direction of the movement. While the potential field principle is particularly attractive because of its elegance

and simplicity, substantial drawbacks have been identified, i.e. local minima (cyclic behavior), no passage between closely spaced obstacles, oscillations in narrow passages ... etc [100].
In the literature, path planning for robots in known and static workspaces has been studied extensively over the last two decades [101]. Path planning for robot navigation in unknown workspaces has been studied less frequently. Kavraki et. al [102] propose the *Probabilistic Roadmaps Method* to generate collision-free paths in a very short time (once that the roadmap has been constructed). However, when generating the paths, no optimization criteria are taken into consideration. In [103] a technique based on the evolutionary computation is proposed to produce a global path optimization with limited computation resources. In [104] the shortest paths problem is formulated as a Markov decision process. A fast replanning method called *D* Lite* for goal-directed navigation is presented in [105]. All these methods have three disadvantages: first, a path-planning algorithm is complete; if it finds a path whenever one exists and reports none exists otherwise. However, achieving this "completeness" is often computationally expensive and grows quickly with the number of locations to be visited. Second, the entire graph representation must be known at all times. Third, the graph topology is assumed to be static, i.e. the calculated trajectory is optimal only, as long as no external perturbations change the connectivity of the graph.

In this chapter, the concept of neural fields will be used to generate the robot behavior over time. We develope a framework to navigate a mobile robot to its goal in unknown environments without any collisions with static or moving obstacles [106]. Furthermore, through their competitive dynamics we optimize the target path through intermediate home-bases. Finally, we design a solution for the problem of moving multiple robots in formation [107]. The objective is to acquire a target, avoid obstacles and keep a geometric configuration at the same time. Simulation results are presented.

7.2 Neural Fields

The Dynamic Approach to behavior generation uses the theory of nonlinear dynamical systems. This theory has proven to be an elegant and easy to generate robot behavior [22, 23, 24, 25]. The so-called *Dynamic Approach* invented by Schöner in 1995 [21] provides a framework to design differential equations for so-called *behavior variables* the solution of which generates

the robot's behavior. Usually, these variables directly parameterize the elementary behavior to be generated. There are cases, however, for which the behavioral variable need a more general form. For example, a behavioral variable can have multiple values or even no value at all. In those cases, it is necessary to express it by a continuous function. Neural fields can represent such variable. Originally, these fields were proposed by Amari [108] as models of the neurophysiology of cortical processes. They are equivalent to continuous recurrent neural networks, in which units are laterally coupled through an interaction kernel and receive external inputs. In [109], neural fields are used to recognize complex motion patterns. In [110], they are used for the intelligent cruise control. In robot control, neural fields were used for target-acquisition in presence of static obstacles, and for manipulator control [111].

The field equation of a one-dimensional neural field is given by:

$$\tau \dot{u}(\varphi, t) = -u(\varphi, t) + S(\varphi, t) + h + \int_{-\infty}^{+\infty} w(\varphi, \acute{\varphi}) f(u(\acute{\varphi}, t) d\acute{\varphi} \tag{7.1}$$

where $u(\varphi, t)$ is the field excitation at time t ($t \geq 0$) at the position $\varphi \in R$.

The temporal derivative of the excitation is defined by

$$\dot{u}(\varphi, t) = \frac{\partial u(\varphi, t)}{\partial t} \tag{7.2}$$

The excitation $u(\varphi, t)$ of the field varies with the time constant τ with $\tau \in R^+$. The constant h defines the pre-activation of the field, and $f(u)$ is the local activation function. Usually, f is chosen as a step-function:

$$f(u) = \begin{cases} 1, & u \geq 0 \\ 0 & u < 0 \end{cases} \tag{7.3}$$

The stimulus $S(\varphi, t) \in R$ represents the input of the field and varies with time. A nonlinear interaction between the excitation $u(\varphi)$ at the position φ and its neighboring positions is achieved by the convolution of an interaction kernel $w(\varphi, \acute{\varphi})$. The function $w(.)$ is, usually, chosen as a Mexican hat function (Figure 7.1). With this shape, excitatory connections dominate for proximate units, and inhibitory connections dominate at greater distances. The structure of the field is specified by this function; when $w(\varphi, \acute{\varphi})$ depends only on $(\varphi - \acute{\varphi})$ then the field is homogenous.

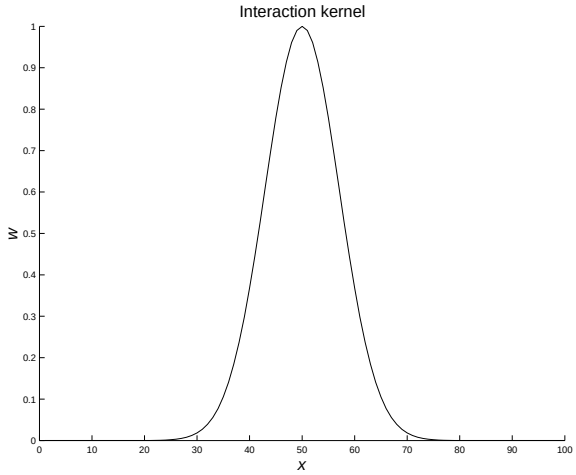


Figure 7.1: Weihting function $w(x)$ of a lateral-inhibition

7.3 Dynamical Properties of Neural Fields

Depending on the parameter h and the form of S, f and w, the activation dynamics (7.1) can have different types of solutions.

7.3.1 Equilibrium Solutions in the Absence of Inputs

Equilibrium solutions mean that $\frac{\partial u}{\partial t} = 0$. In the absence of an external stimulus ($S(\varphi, t) = 0, \forall\varphi$), the equilibrium solutions satisfy

$$u(\varphi) = \int_{-\infty}^{+\infty} w(\varphi - \acute{\varphi}) f[u(\acute{\varphi})] d\acute{\varphi} + h \tag{7.4}$$

Definition 1 *Let the set $R[u] = \{\varphi | u(\varphi) > 0\}$ of all exited units at place φ.*

- *Equilibrium $u(\varphi)$ satisfying $R[u] = \emptyset$, i.e $u(\varphi) \leq 0, \forall\varphi$, is called "$\emptyset$-solution".*
- *Equilibrium $u(\varphi)$ satisfying $R[u] = (-\infty, +\infty)$, i.e the whole field is excited, is called "∞-solution".*
- *Equilibrium $u(\varphi)$ satisfying $R[u] = [a_1, a_2]$, i.e a localized excitation from position a_1 to position a_2, is called "a-solution". If only one a-solution exists, this solution is called also a "single-peak" or "mono-modal" solution.*

The correct choice of the field parameters enables the existence of a single-peak solution.
Let

$$W(x) = \int_0^x w(y)dy \tag{7.5}$$

In order to specify some properties of the field, we put

$$W_m = \max W(x), \forall x > 0 \tag{7.6}$$

and

$$W_\infty = \lim_{x\to\infty} W(x) \tag{7.7}$$

Under conditions that the output function f is a step function, and the interaction kernel w is symmetric ($w(x) = w(-x)$), then the following theorems give the conditions for the existence of equilibrium $\emptyset$-solution, ∞-solution, and a-solution, and their stability conditions.

Theorem 1 *In the abscence of inputs:*

1. *There exists a $\emptyset$-solution if and only if $h < 0$.*
2. *There exists an ∞-solution if and only if $2W_\infty > -h$.*
3. *There exists an a-solution (a local excitation of length a) if and only if $h < 0$ and $a > 0$ satisfies $W(a) + h = 0$.*

Theorem 2 *In agreement with Theorem 1, the a-solution is asymptotic stable if $\frac{dW(a)}{da} < 0$, and unstable if $\frac{dW(a)}{da} > 0$.*

The proof of theorems (1) and (2) and a complete list of equilibrium solutions in relation with the function $W(x)$ are given in [108].

7.3.2 Response to Stationary Input Stimulus

In this section, we consider the field in a-solution. When an input of a stimulus $S(\varphi, t)$ is very large compared with the within-field cooperative interaction, it will dominate the solution. As a result, a single-peak will be stabilized by interaction, even if the stimulus is removed.
Amari studied the effect of a small variation of the stimulus $\varepsilon\tilde{S}(\varphi, t)$, where ε is a small number. This variation results to a displacement and deformation of the a-solution, according to the theorem 3 (for demonstrations see [108]).

Theorem 3 *Consider that the field is in a localized stable equilibrium excitation (a-solution) produced by an external stimulus $S(\varphi, t) = S_0$. Let a_0 be the length of the excited region, and $\varphi_1(t)$ and $\varphi_2(t)$ be the boundaries of the excited region. After variation on the stimulus $S(\varphi, t) = S0 + \varepsilon\tilde{S}(\varphi, t)$, where $1 > \varepsilon > 0$ and $\tilde{S}$ is differentiable, the dynamic of the center of the excited region is given by:*

$$\frac{1}{2}\frac{d(\varphi_1 + \varphi_2)}{dt} = \frac{\varepsilon}{2\tau c}[S(\varphi_2) - S(\varphi_1)] \tag{7.8}$$

If we assume that the length of the excited region is $a(t) = a_0 + \varepsilon a_1 + O(\varepsilon^2)$, and a_1 is differentiable, then the change of length satisfies:

$$\frac{da_1}{dt} = \frac{1}{\tau c}[2w(a_0)a_1 + S(\varphi_1) + S(\varphi_2)] \tag{7.9}$$

where c is the gradient of the $u(\varphi)$ at the boundary.

From equation (7.8), the excited region moves in the direction of increasing stimulus, searching for the maximum of $S(\varphi)$. At the same time, the length of the excited region changes slightly, according to (7.9).

7.4 Behavior control with neural fields

In this section, we show how to use neural fields to control a robot's planar movements regarding:

. Target-Acquisition: moving towards a given target point.

. Obstacle Avoidance: moving while avoiding obstacles

. Corridor Following: moving along corridors in the absence of obstacles.

. Door Passing: passing a door to traverse between two rooms, from a corridor to a room, or vice versa.

7.4.1 Control Design

First, we choose the robot's heading φ relative to a world-fixed reference direction as a behavioral variable. Hence, the neural field has to encode

angles from $-\pi$ to $+\pi$. By means of a codebook we use N discrete directions.

For numerical reasons equation (7.1) is discretised:

$$\tau \dot{u}(i,t) = -u(i,t) + S(i,t) + h + \sum_{j=1}^{N} w(i,j) f[u(j,t)] \tag{7.10}$$

where τ ($\tau \in R^+$) is the time constant. The constant h defines the pre-activation of the field, and $f(u)$ is the local activation function. $w(i,j)$ is the interaction kernel between neurons i and j. $S(i,t)$ is the external stimulus at the neuron i. Since the field is periodic, the interaction kernel has to be periodic too:

$$w(i,j) = k_w e^{-\sigma_w (i-j)^2} - H \tag{7.11}$$

where the parameter σ_w and k_w are used to adjust the range of excitation and its amplitude, respectively. The global inhibition H allows only one localized peak on the field.
After the stabilization of the field, the most activated neuron decodes the direction to be executed by a robot:

$$\varphi_F = arg_{\max}\{u(i) | i \in [1, N]\} \tag{7.12}$$

Field Stimulus

Before selecting an appropriate robot direction, the neural field needs some necessary information (stimulus). Based on the tasks cited above, the stimulus is determined according to two stimulus-functions. These functions describe:

1 the target direction: This stimulus is designed excitatory, showing an attraction towards the target direction. It is chosen as:

$$S_T(i,t) = C_{T1} - C_{T2} |i - i_T(t)| \tag{7.13}$$

where C_{T1}, C_{T2} are constants positive, and $i_T(t)$ is the field position, equivalent to the target direction at time t.

2 directions to obstacles $\{S_{Ol}(i,t) : l \in [1, N_{Obst}]\}$, where N_{Obst} are the number of obstacles detected by the robot sensors. However, the stimulus consider only obstacles, which their distances d_{Ol} to the robot are less

than a threshold d_{Th}. This stimulus must be inhibitory, since obstacles collision must be avoided. It is chosen as a Mexican Hat function centered at the direction of an obstacle.

$$S_{Ol}(i,t) = C_O e^{-\sigma_O (i-i_l)^2} \tag{7.14}$$

where C_O and σ_O are positive constants. σ_O defines the range of inhibition of an obstacle. In practical situations, this parameter is tuned regarding the radius of the robot and the obstacles. i_l reflects the direction of the obstacle l at time t.

The contributions of different stimulus define the state of the field. Thus, the stimulus of the field at time t is determined by

$$S(i,t) = S_T(i,t) - \sum_{l=1}^{N_{Obst}} g(d_{Ol}) S_{Ol}(i,t) \tag{7.15}$$

where g is a step function:

$$g = \begin{cases} 1, & d_{Ol} < d_{Th} \\ 0 & d_{Ol} \geq d_{Th} \end{cases} \tag{7.16}$$

Dynamics of Speed

In a free obstacle situation, the robot moves with its maximum speed $V_{\max}$, and slows down when it approches a target. This velocity dynamics can be chosen as:

$$V_T(t) = V_{\max}(1 - e^{-\sigma_v d_T}) \tag{7.17}$$

where σ_v is a positive constant tuned in a relation with the acceleration capability of the robot. d_T represents the distance between the robot and the target at time t.

Close to obstacles, the robot needs also to be slowed down. In case of many obstacles, the nearest obstacle in the robot direction is considered. This dynamics can be chosen as:

$$V_O(t) = C_O(1 - g(d_{no}) e^{-\sigma_{Ov}(\varphi_F - i_{nO})^2}) \tag{7.18}$$

where C_O and σ_{Ov} are positive constant, i_{nO} is the direction of the nearest obstacle, which is on the path way of the robot. d_{nO} is its relative distance.

The final dynamics of the velocity is the contribution of (7.17) and (7.18). Furthermore, it is also considered when no appropriate direction can be selected, for example when the robot is completely surrounded by obstacles. In this case the robot must stop until the environmental situation changes. Thus, the robot velocity that satisfies the above design creteria is the following:

$$V(t) = \begin{cases} V_T(t)V_O(t), & \varphi_F > 0 \\ 0 & \varphi_F \leq 0 \end{cases} \tag{7.19}$$

7.4.2 Results

In order to update the field stimulus, we assume a 360^o sensor. Its range is assumed to be a circle with a radius equal to 1. On the field we chose $N = 60$ neurons, which means that each direction N decodes a step of 6^o. The position of a target is given in the spatial coordinates (X_{tar}, Y_{tar}) relative to the origin of the robot's map. Form the actual position of the robot (X_r, Y_r); the angle of the direction to the target is calculated:

$$\varphi_{tar} = \arctan(\frac{Y_{tar} - Y_r}{X_{tar} - X_r}) \tag{7.20}$$

Target-acquisition with Obstacle avoidance

The behavior *Target-acquisition* is expected to align the robot's heading with the direction of the target. During movement, the behavior *obstacle-avoidance* is expected to bring the robot away from the nearby obstacles. We will illustrate this with two simulation examples: Collision avoidance for static and moving obstacles. Figure 7.2.(e) shows the path movement of the robot from its initial position to the target position. In the first 14 time steps, both, the stimulus (Figure 7.2.b) as well as the field activation (Figure 7.2.a) are unimodal, since no obstacles are detected and the stimulus contains only the target entries. In the following time steps, the stimulus contains also obstacle entries. Therefore, it becomes bimodal. By contrast, on the field activation, the peak follows the optimum position resulted from the combination of *target-acquisition* and *obstacle-avoidance* behaviors. It provides now the appropriate heading direction towards the target with obstacle avoidance (Figure 7.2.d). Furthermore, according to (7.19), the robot moves with its maximum linear velocity when no obstacles are detected, and slowed down when it is near the obstacle or approaches the target (Figure 7.2.c).

For most real-world applications, a mobile robot has to move within a dynamic environment. In this context, the problem is how to reach a target in the presence of dynamically moving obstacles. As seen above, when an obstacle is static, avoidance is accomplished by measuring its relative position. When an obstacle is moving, on the other hand, collision avoidance is harder because the robot has to detect not only the position but also the direction of the moving obstacle. In this experiment, the objective is to navigate the mobile robot to its goal in an unknown environment without any collisions with moving obstacles. During the first 7 time steps, both, the stimulus (Figure 7.3.b) as well as the field activation (Figure 7.3.a) are unimodal, since the stimulus contains only the target entries. In the following time steps, the stimulus contains the obstacle 1 entries. On the field activation, the peak changes its position according to the stimulus. Thus, the robot changes its direction (Figure 7.3.d) and slows down its speed (Figure 7.3.c). At time 16, the entries of the target becomes stronger than those of the obstacle. The peak change its position relative to the direction of the target. At time 18 the stimulus contains entries of obstacle 2. Again, the robot behaves with the same manner until it reaches the target. Figure 7.3.(e) shows the entire path movement of the robot.

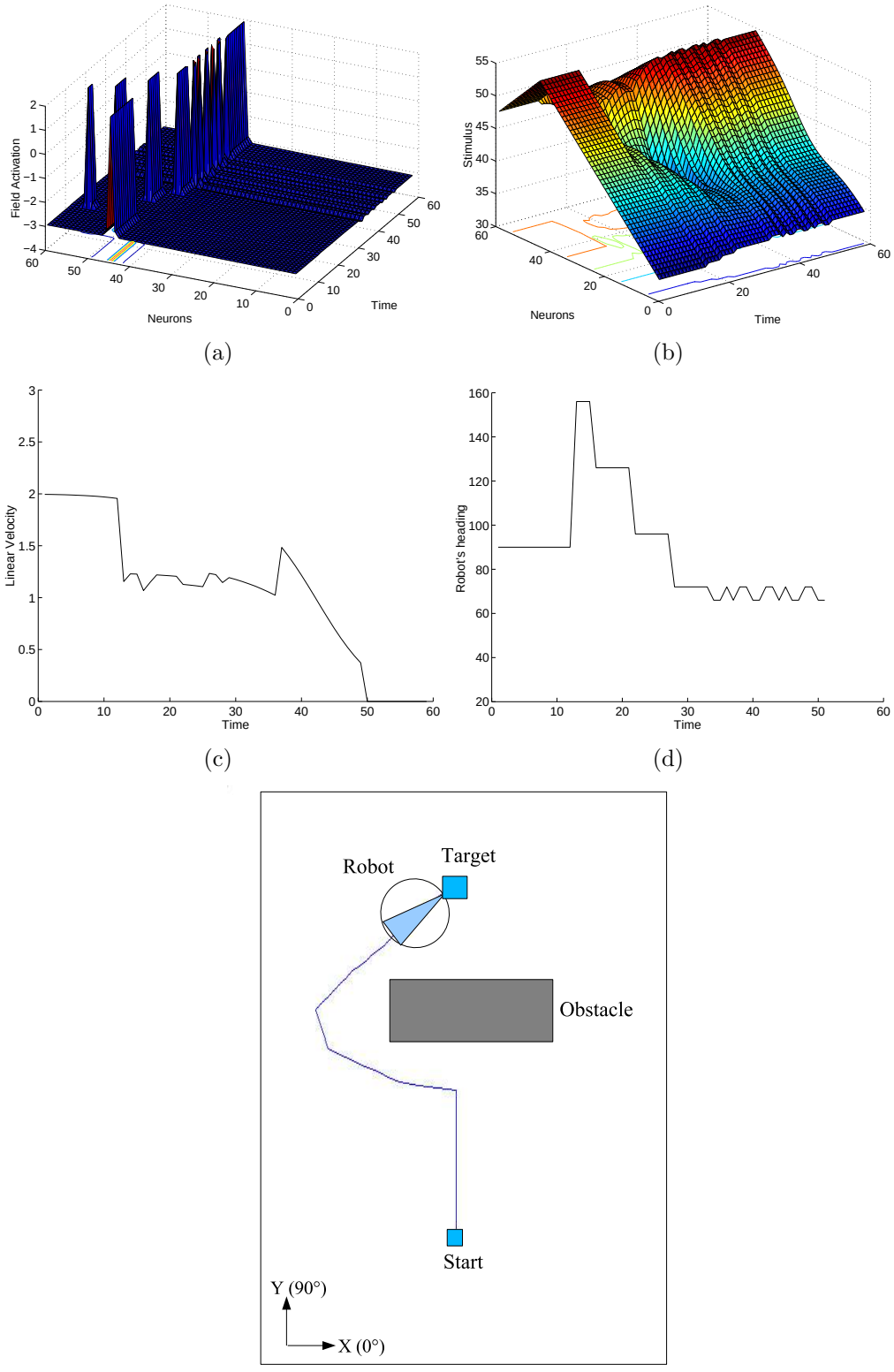


Figure 7.2: Target acquisition with Obstacle avoidance

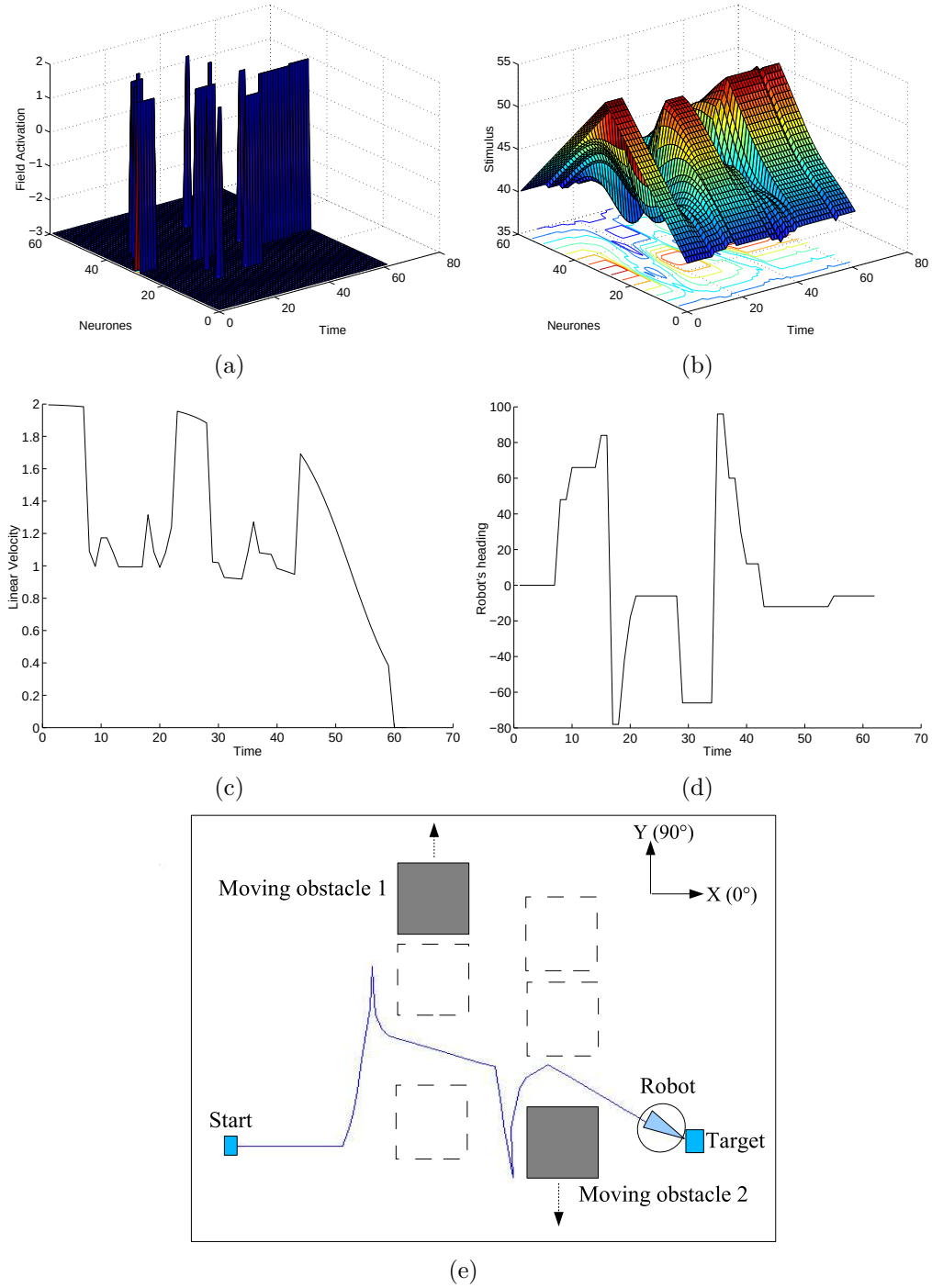


Figure 7.3: Collision Avoidance for Moving Obstacles. The dotted lines in (e) represent the old positions of the obstacles.

Corridor Following

The behavior *Corridor-Following* navigates the robot along an empty corridor. Two experiments are presented: in the first, the initial position of the robot is supposed to be in the middle between the corridor walls, as well as the target point. This is the simplest case, since it can be considered as a target acquisition with no obstacles. During the whole experiment, the field has only one peak on the heading direction 0^o (Figure 7.4.a). This behavior is expected, since the stimulus contains only the target entries (Figure 7.4.b). Thus, the robot moves with the heading angle 0^o until it reaches the target (Figure 7.4.e), and its velocity depends only on the distance relative to the target (Figure 7.4.c). In the second experiment, the robot starts from a position near a wall (Figure 7.5.e). In this case, the field is expected to guide the robot towards the target on the center of the corridor, with wall avoidance. In the first 14 time steps, the stimulus contains obstacle entries (Figure 7.5.b). Therefore, the field's peak moves to an optimum provided by the stimulus (Figure 7.5.a). The robot moves with negative heading angles, until the behavior *target-acquisition* dominates the stimulus entries (Figure 7.5.d).

Door Passing

The behavior *Door-Passing* is supposed to navigate the robot to a target, but through a door. This is in principal the same behavior *target-acquisition* with *obstacle-avoidance*. However, the door has to be large enough, so that at the door, the stimulus entries will be dominated by the target acquisition. This is necessary to lead the robot through the door. Figure (7.6) shows the experiment result.

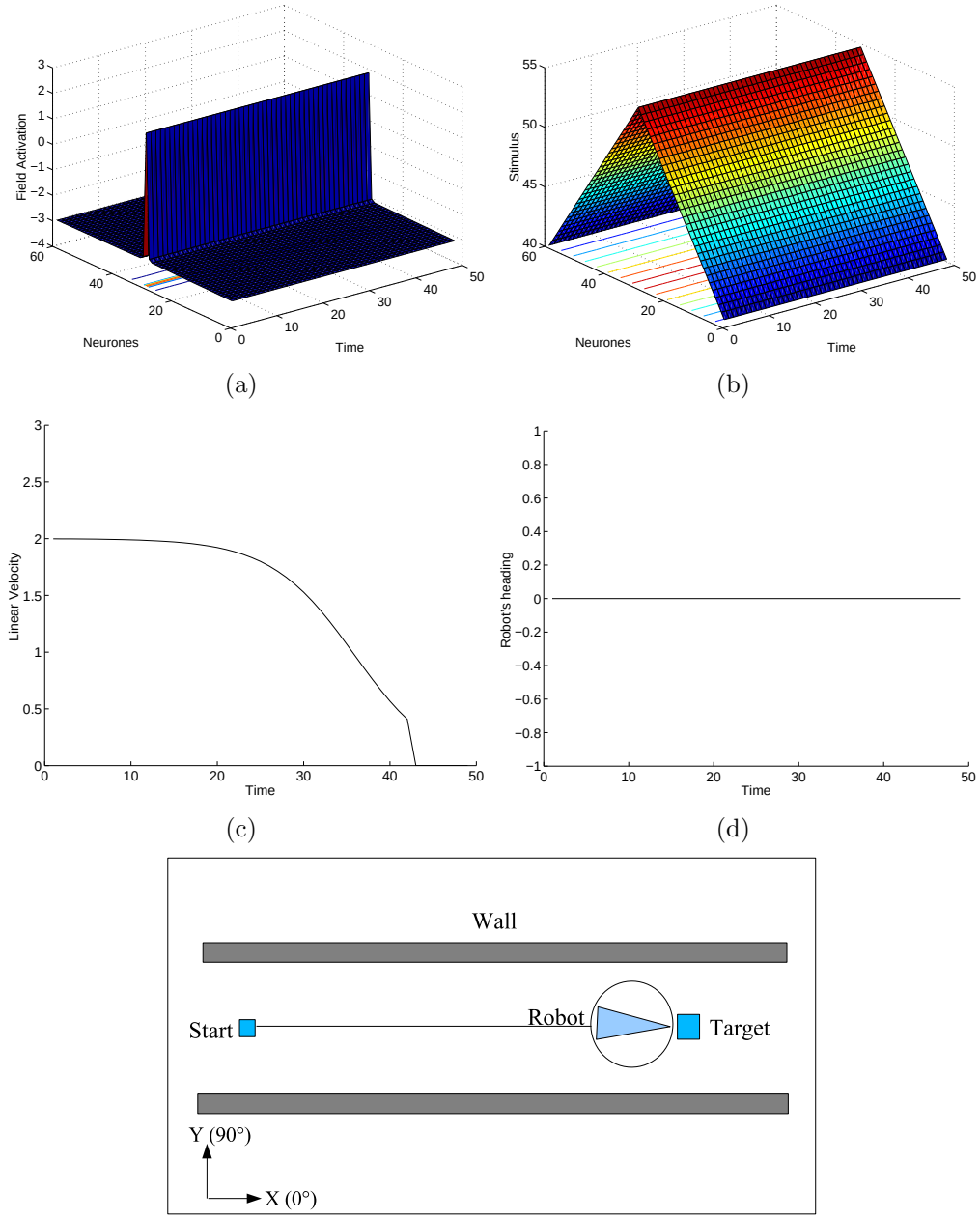


Figure 7.4: Corridor Following: experiment 1

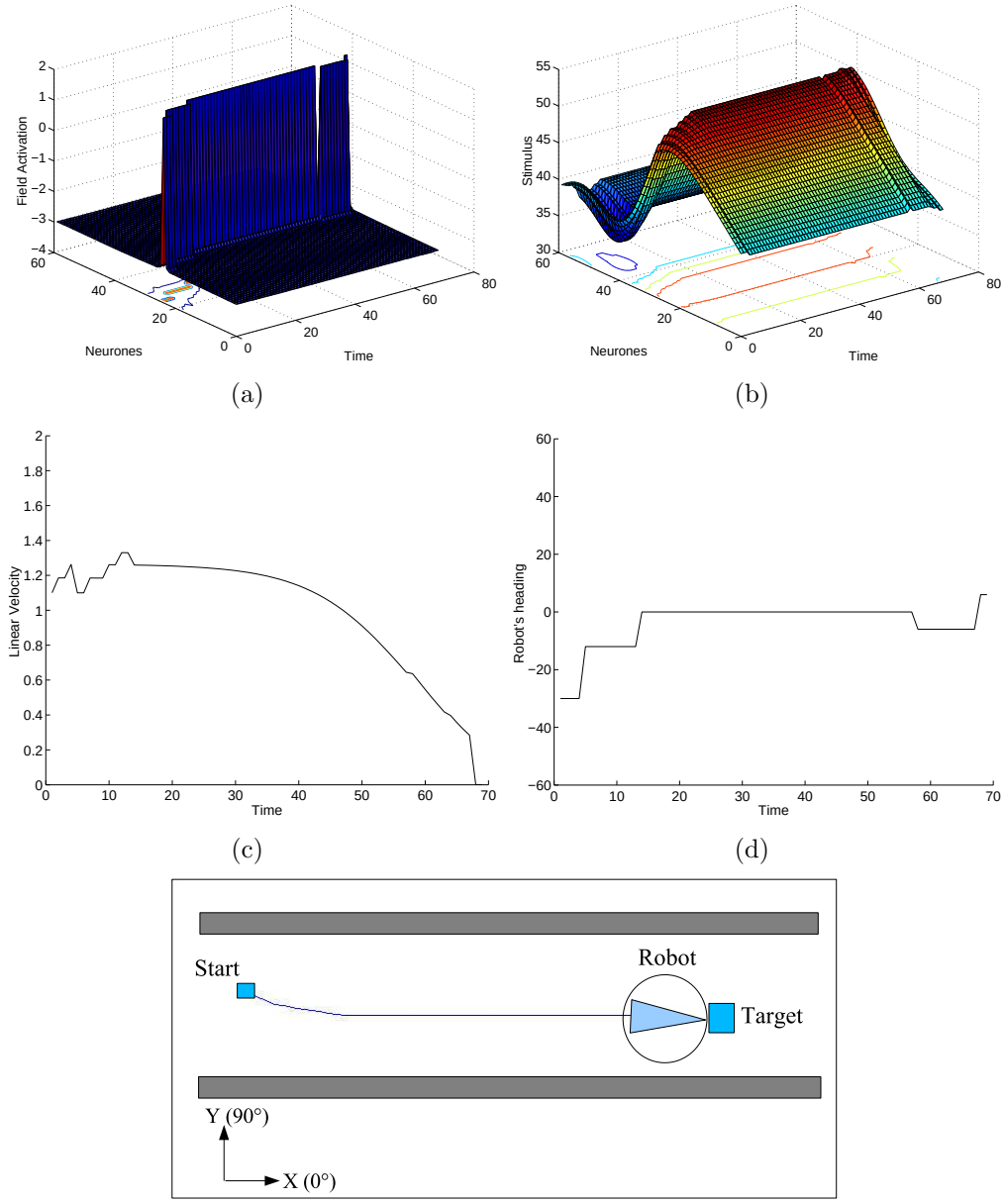
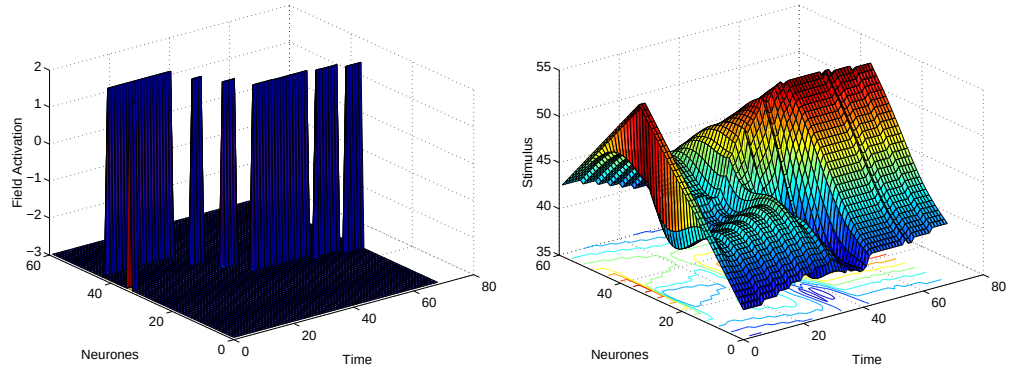
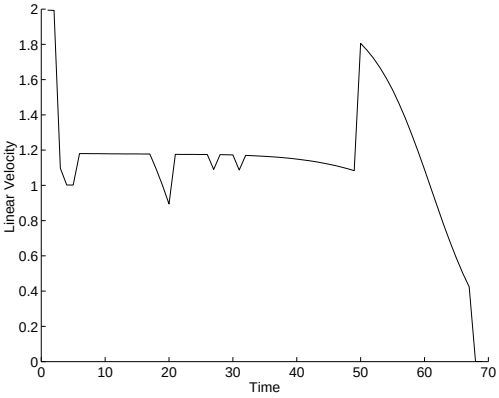


Figure 7.5: Corridor Following: experiment 2

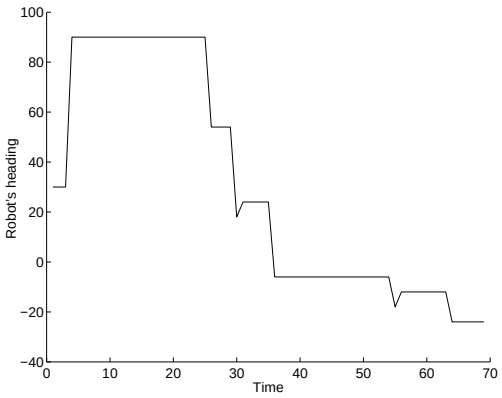


(a)

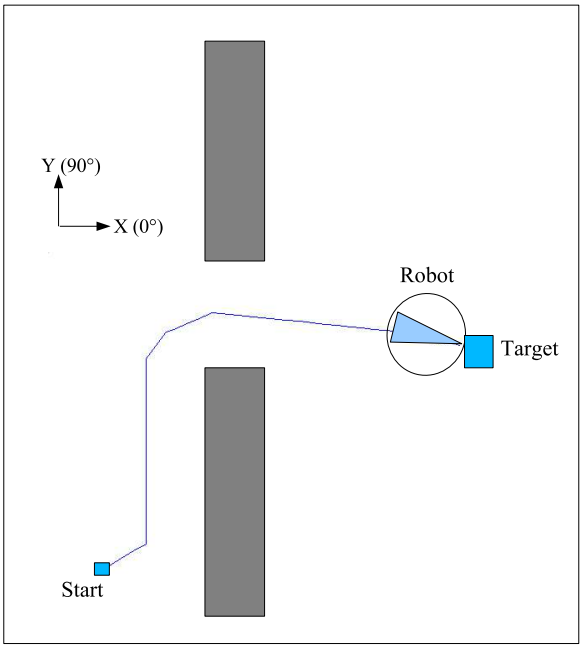
(b)



(c)



(d)



(e)

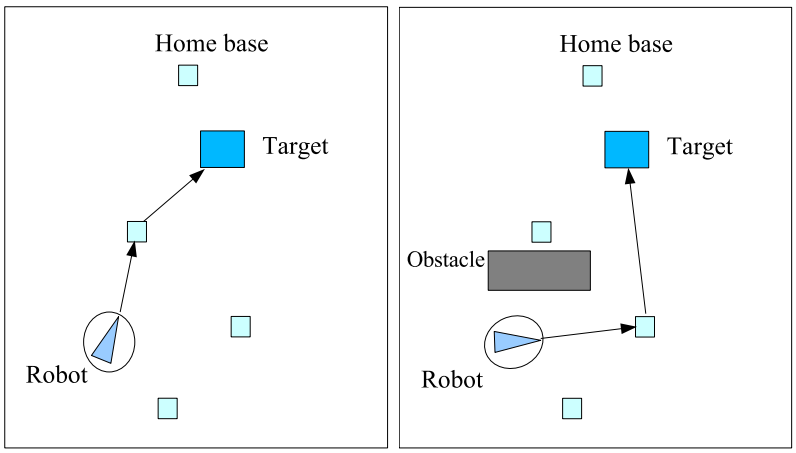
Figure 7.6: Door Passing

7.5 Competitive Dynamics for Home-bases Acquisition

Eilman and Golani [112] show by experiments that when rats are brought in a novel environment, they move alternately between stop and go. The place in which they stay for the longest time is defined as the rat's home-base. In addition, it was observed, that when they want to reach a target, they prefer going through these home-bases instead of going through novel environments. These home-bases are typically installed only in areas, which are meaningful for them (e.g food places). In behavior control, this can be considered as an optimisation problem of planning reliable landmark-based robot navigation. A necessary condition for this optimization is that the generated path should be the shortest path through the maximum intermediate home-bases. Naturally, this must be not much longer than the direct way to the target.

In this section, we adopt the concept of neural fields to optimize the target path through home-bases. The idea is to use an extra one-dimensional neural field, we call it *sub-target neural field*, that expresses, whether a home-base is the next sub-target or not. A Peak will decode the direction to a sub-target whenever one, or no peak otherwise. Following this method, the entire optimal trajectory from the start to the main target will be not necessary to be known, but only local information about the next sub-target. This implies also that the optimal decisions are not static: a change in the accessibility of a home-base leads to change the decision on which optimal home-base to visit next. To produce an optimal path planning, the sub-target neural field should produce a peak solution based on two requirements: a sub-target must be the nearest home-base to the robot and on the way to the main target, i.e. the shortest path to the main target (Figure 7.7.a), and flexibility: when for instance an obstacle forces the robot to make a detour, the path should be changed to another home-base (Figure 7.7.b). This makes a re-calculation of the optimization not necessary.

Although the local decisions made by the two neural fields, the trajectory generated through all sub-targets will be the global optimal one, if no changes in the environment occur.



(a) (b)

Figure 7.7: Sub-targets Acquisition

7.5.1 Sub-target Neural Field

This field has almost the same characteristics as the navigation field (7.10). It encodes home-bases angles from $-\pi$ to π :

$$\tau \dot{u}_{ST}(i,t) = -u_{ST}(i,t) + S_{ST}(i,t) + h + \sum_{j=1}^{N} w(i,j) f[u_{ST}(j,t)] \qquad (7.21)$$

where $u_{ST}(i)$ is the field activation at the position i. A nonlinear interaction with its neighboring j positions is achieved by the convolution of an interaction kernel $w(i,j)$. τ ($\tau \in R^{+}$) is the time constant. The constant h defines the pre-activation of the field, and $f(u_{ST})$ is the local activation function. $S_{ST}(i,t)$ is the external stimulus at the neuron i.

Home-bases Stimulus

As for navigation, the neural field (7.21) needs some necessary information (stimulus) about the home-bases locations. A home-base becomes a sub-target if it fulfills the requirement of being the nearest one to the robot and on the way to the main target. Thus, the stimulus is based on the location of the home-bases, on the position of the main target and on the current position of the robot. This stimulus is designed excitatory, showing

an attraction towards a sub-target. The stimulus of each home-base k is chosen as:

$$S_{HBk}(i,t) = \begin{cases} C_{HB}(d_T - d_{HBk})e^{-\sigma_{HB}(i-i_{HBk})^2}, & d_{HBk} < d_T \\ 0, & d_{HBk} \geq d_T \end{cases} \tag{7.22}$$

where C_{HB} and σ_{HB} are positive constants, which depends on the field characterstics. d_T and d_{HBk} are the distances of the main target and the home-base k relative to the robot, respectively. The distance d_{HBk} gives an advantage to the home-bases, which lie closer to the robot. However, the term $C_B(d_T - d_{Bk})$ alone is not enough, because the generated trajectory may not be the globally optimal. It may lead to detours, which violate the requirement of shortening the path. To prevent this, a Mexican Hat function ($e^{-\sigma_{ST}(i-i_T)^2}$), centred at the direction of the main target i_T, is used to eliminate the effect of all home bases, which are not on the way to the main target. Furthermore, obstacles must be taken into consideration, in order to change the plan if necessary. The global stimulus of the Sub-target neural field is the contribution of all home bases stimulus. Thus the global stimulus at time t will be:

$$S_{ST}(i,t) = \sum_k S_{HBk}(i,t)e^{-\sigma_{ST}(i-i_T)^2} - \sum_{l=1}^{N_{Obst}} g(d_{Ol})S_{Ol}(i,t) \tag{7.23}$$

7.5.2 Target-acquisition Stimulus

To acquire these sub-targets, the stimulus of the navigation neural field (7.15) must be modified. When a sub-target is localized, the robot should move to it and ignore the attraction of the main target. Thus the influence of the main target is switched off, and the active sub-target errects a peak solution. When an obstacle is detected or no sub-target is active, the main target is acquired. These requirements can be reached by the modification bellow:

$$S(i,t) = (1 - HV(u_{ST}))(S_T(i,t) - \sum_{l=1}^{N_{Obst}} g(d_{Ol})S_{Ol}(i,t)) + HV(u_{ST})S_{ST}(i,t) \tag{7.24}$$

where $HV(.)$ is the heavside function:

$$HV(u_{ST}) = \begin{cases} 1, & u_{ST} > 0 \\ 0, & u_{ST} \leq 0 \end{cases} \tag{7.25}$$

7.5.3 Results

The sub-target neural field has the same number of neurons ($N_{ST} = 60$). We consider that the position of a home-base in the spatial coordinates is (X_{HB}, Y_{HB}). From the actual position of the robot (X_r, Y_r), the angle to a home-base is:

$$\varphi_{HB} = \arctan(\frac{Y_{HB} - Y_r}{X_{HB} - X_r}) \tag{7.26}$$

The main target is acquired by planning optimal sequences through the intermediate home-bases. Figure (7.8) illustrates a first experiment. In the first 20 time steps, the sub-target field selects, through competition, the home-bases 1 and 3 to be intermediate sub-targets to the main target (Figure 7.8. b). In the following time steps, it can be seen that when there is no sub-targets on the target way, the main target is acquired. This result demonstrates that the first requirement of shortening the target path, mentioned earlier, is fulfilled. The second requirement of flexibility is tested in the next simulation. Figure (7.9) shows the case, where the robot should avoid an obstacle. We repeat the same experiment above, but an obstacle is added on the way to the home bases 1 and 3. When the obstacle is detected, the influence of the home base 1 is switched off, and then only the main target is acquired with obstacle avoidance. After this detour, the home-bases 1 and 3 are not any more sub-targets. Instead, the home base 2 has the more advantage to be a sub-target, which is interpreted by switching ON the *sub-target field*. After passing the home-base 2, the main target is acquired.

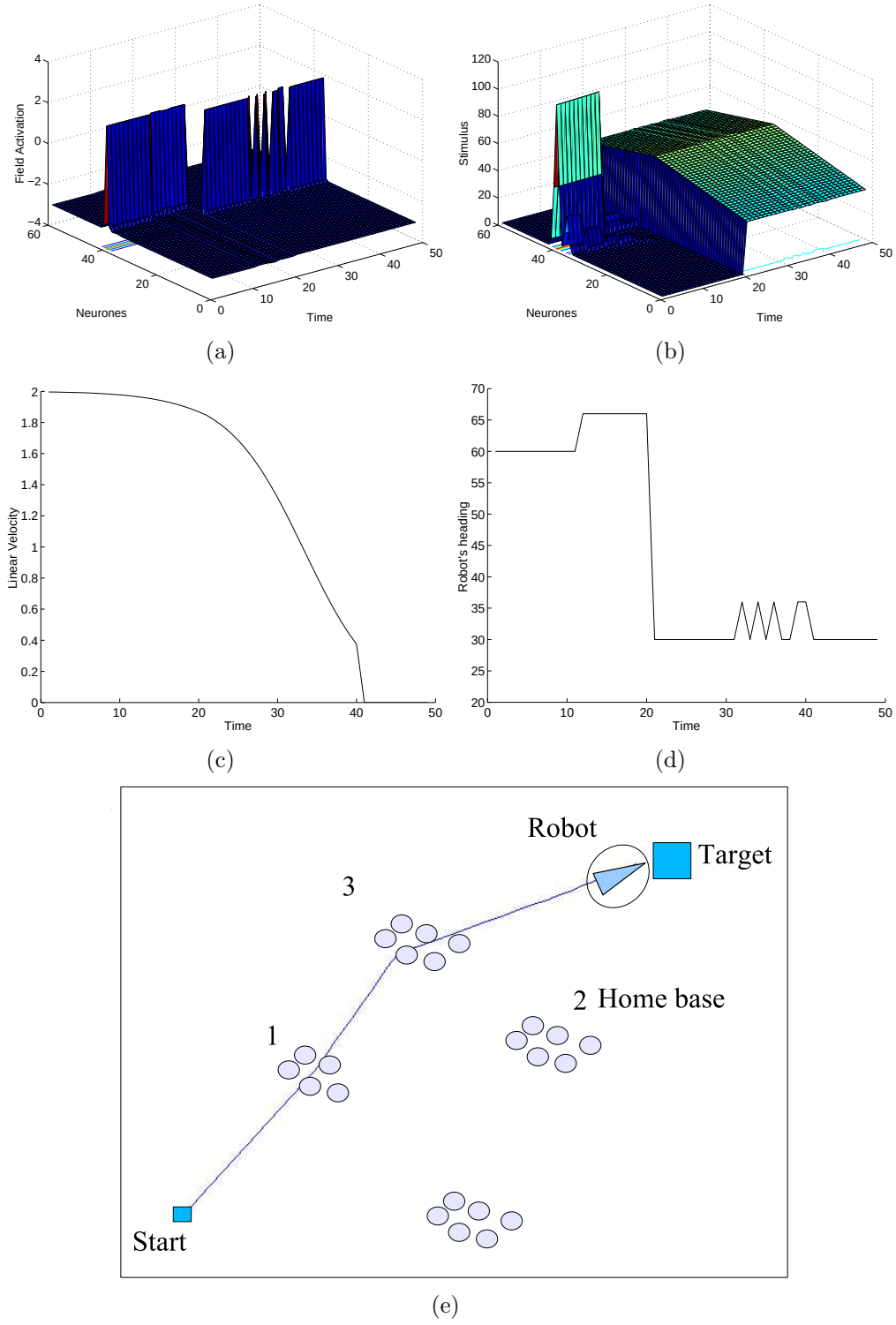


Figure 7.8: Sub-target Acquisition Through Competition

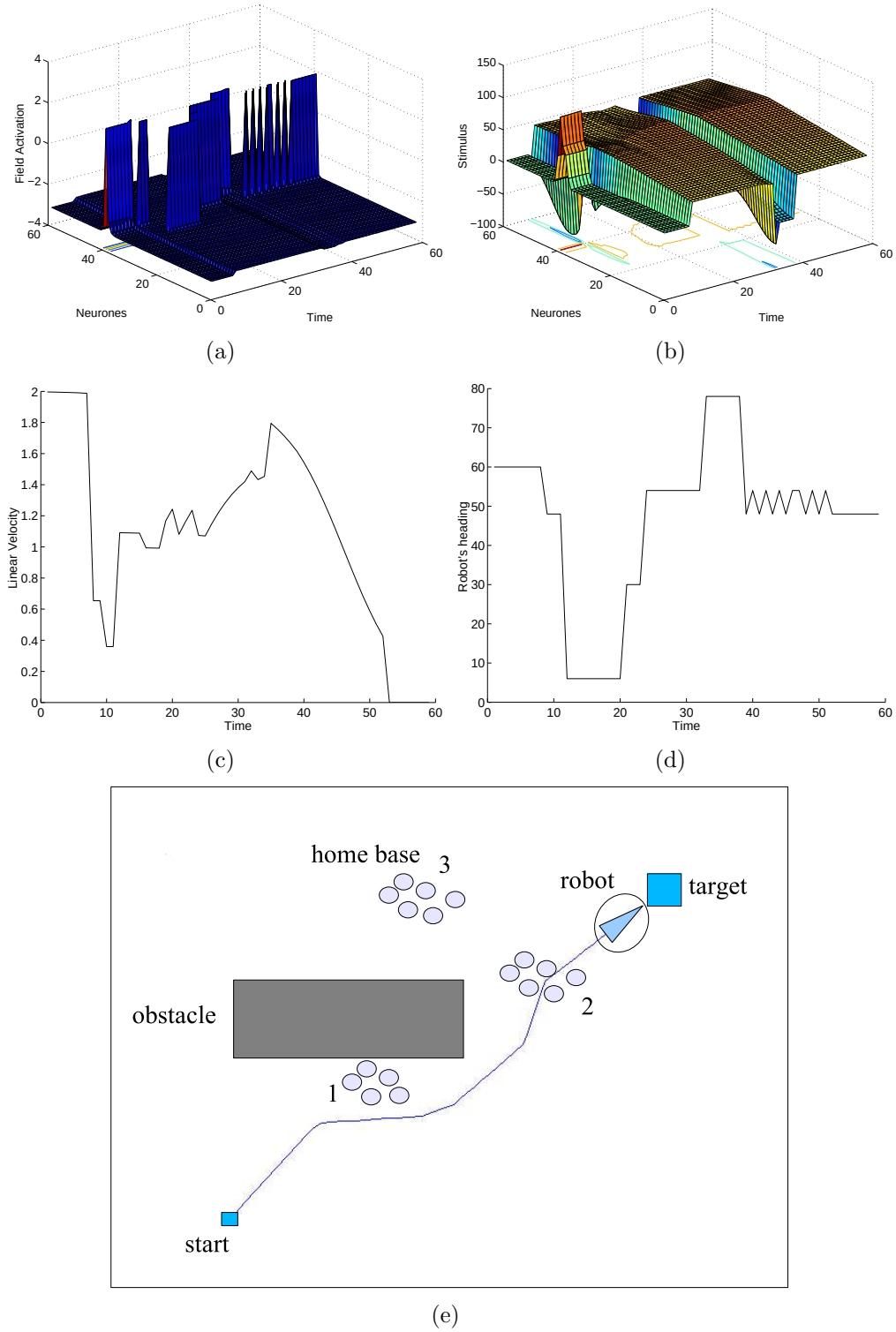


Figure 7.9: Sub-targets Acquisition: Flexibility

7.6 Neural Fields for Multiple Robots Control

Recently, there has been increased research interest in systems composed of multiple mobile robots exhibiting cooperative behavior. Such systems are of interest for several reasons: i) tasks may be too complex (or impossible) for a single robot to accomplish, and ii) using several simple robots can be easier, cheaper, and more flexible than having a single powerful robot for one task. This is inspired from the nature. Living in a group provides animals to combine their sensors to maximize the chance of detecting predators, or searching for food.
A key problem in cooperative robotics is to maintain a certain geometric configuration during movement. The reason is that there are many interesting tasks that require multiple robots to coordinate their movements for example box-pushing [113], and movement of a team of military robotic vehicles as a scout unit [114]). There are two main ways of approaching this problem: model-based control, and behavior-based control. In the first, the objective is to build a model for the team of robots and the desired task, and develop a framework to optimise their performance [115]. This method has a limit success when the robots behave in a dynamic and unknown workspace. By contrast, in behavior-based control no model is required. The control system is in a form of reactive behaviors to the current state of the environment [114].

In this section we investigate how neural fields can produce an elegant solution for the problem of moving in formation. The objective is to acquire a target, avoid obstacles and keep a geometric configuration at the same time. Several formations for a team of three robots are considered (Figure 7.10):

- *line*: robots move line-abreast,
- *column*: robots move one after the other,
- *triangle*: robots move in a triangle formation.

The strategy is to have a leader (robot 1) guiding followers (robots 2 and 3). The robots should avoid obstacles and collisions among them. We assume that the robots are identical, and dont fail during travel.

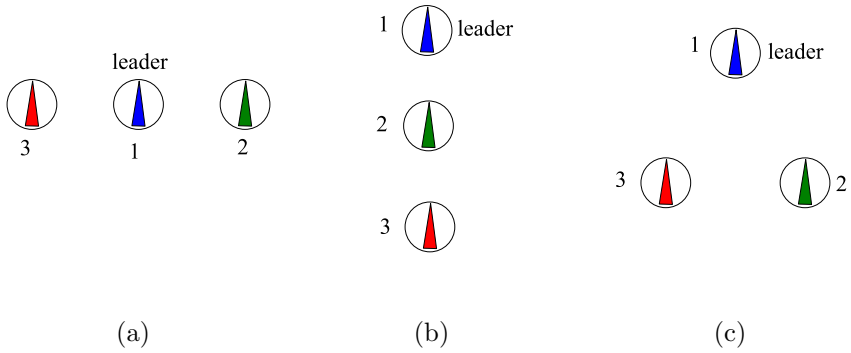


Figure 7.10: Geometric Configurations: a) line, b) column, c) triangle

7.6.1 Control Design

Each robot 1, 2, and 3 has its own neural field model. We choose the robots' headings φ_k ($k \in \{1, 2, 3\}$) relative to a world-fixed reference direction as behavioral variables. Hence, each neural field has to encode angles from $-\pi$ to $+\pi$. By means of a codebook we use N discrete directions.

The three neural field models will be described by:

$$\tau \dot{u}_k(i,t) = -u_k(i,t) + \sum_{j=1}^{N} w(i,j) f[u_k(j,t)] + S_k(i,t) + h \tag{7.27}$$

where $k \in \{1, 2, 3\}$.

Each interaction kernel is chosen as:

$$w(i,j) = k_w e^{-\sigma_w (i-j)^2} - H \tag{7.28}$$

where the parameter σ_w and k_w are used to adjust the range of excitation and its amplitude, respectively. The global inhibition H allows only one localized peak on the field.
After the stabilization of the field, the most activated neuron decodes the direction to be executed by a robot:

$$\varphi_F = arg_{\max}\{u(i) | i \in [1, N]\} \tag{7.29}$$

Before selecting an appropriate direction of a robot, each neural field needs some necessary information (stimulus).

7.6.2 Field Stimulus

Based on the tasks described earlier, the stimulus is determined according to two stimulus-functions. These functions describe:

1 the direction towards the target. This stimulus is designed excitatory, showing an attraction to the desired direction of each robot.
The target of the leader is the "main target". Thus, its stimulus can be chosen as:

$$S_{Tleader}(i,t) = C_{T1} - C_{T2}|i - i_{main}(t)| \tag{7.30}$$

where C_{T1}, C_{T2} are constants positive, and $i_{main}(t)$ is the field position, equivalent to the main target direction at time t.
The followers are attracted by the required configuration relative to the leader. In this case each follower has the target stimulus :

$$S_{Tk}(i,t) = C_{T1k} - C_{T2k}|i - i_{Tk}(t)| \tag{7.31}$$

where C_{T1k}, C_{T2k} are constants positive, and $i_{Tk}(t)$ is the field position, equivalent to the direction of the required position of the robot k ($k \in \{2,3\}$) at time t.

2 the directions to obstacles $\{S_{Ol}(i,t) : l \in [1, N_{Obst}]\}$, where N_{Obst} are the number of obstacles detected by the robot sensors. This stimulus must be inhibitory, since obstacles collision must be avoided. It is chosen as a Mexican Hat function centered at the direction of an obstacle.

$$S_{Ol}(i,t) = C_O e^{-\sigma_O (i - i_l)^2} \tag{7.32}$$

where C_O and σ_O are positive constants. σ_O defines the range of inhibition of an obstacle. In practical situations, this parameter is tuned regarding the radius of the robot and the obstacles. i_l reflects the direction of the obstacle l at time t. Obstacles here include also other robots, since they should avoid collision among tem. However, the stimulus consider only obstacles, which their distances d_{Ol} to the robot are less than a threshold d_{Th}. Thus, the global obstacle stimulus is determined by:

$$S_O(i,t) = \sum_{l=1}^{N_{Obst}} g(d_{Ol}) S_{Ol}(i,t) \tag{7.33}$$

where g is a step function:

$$g(d) = \begin{cases} 1, & d < d_{Th} \\ 0 & d \geq d_{Th} \end{cases} \tag{7.34}$$

Using the step function g, only obstacles , which their distances d_{Ol} to the robot are less than a threshold d_{Th}, are considered.

The contributions of different stimulus define the global stimulus for each neural field. For the leader neural field, the stimulus at time t is determined by

$$S_{leader}(i,t) = S_{Tleader}(i,t) - C_M S_O(i,t) \tag{7.35}$$

For the followers, the stimulus is:

$$S_k(i,t) = S_{Tk}(i,t) - C_M S_O(i,t) \tag{7.36}$$

where $k \in \{2,3\}$, and C_M is a constant positive.

7.6.3 Formation Control

For each configuration, two tasks, *formation-speed* and *formation-steer* run simultaneously to maintain each robot in its desired position.

formation-speed

In a free obstacle situation, the leader moves with its maximum speed $V_{\max}$, and slows down when it approches a target. This dynamics can be chosen as:

$$V_T(t) = V_{\max}(1 - e^{-\sigma_v d_T}) \tag{7.37}$$

where σ_v is a constant tuned in a relation with the acceleration capability of the robot, and d_T represents the distance between the robot and the target at time t. When it approaches obstacles, its velocity must be reduced. This dynamics can be chosen as:

$$V_O(t) = C_O(1 - g(d_{no})e^{-\sigma_{Ov}(\varphi_F - i_{nO})^2}) \tag{7.38}$$

where C_O and σ_{Ov} are constant. i_{nO} is the nearest obstacle direction to the robot movement direction, and d_{nO} is its distance relative to the robot. The final dynamics of the leader velocity is the contribution of (7.37) and

(7.38). It is also considered when no appropriate direction can be selected, for example when the robot is completely surrounded by obstacles. In this case the robot must stop until the environmental situation changes. Thus, the global leader velocity that satisfies the above design creteria is the following:

$$V_{leader}(t) = \begin{cases} V_T(t)V_O(t), & \varphi_F > 0 \\ 0 & \varphi_F \leq 0 \end{cases} \tag{7.39}$$

The following steps summarize how a follower selects its speed:

. If a follower is behind its desired position, it should speed up.

. If a follower is in front of its desired position, it should slow down.

In presence of obstacles, the speed is slowed down, depending on the distance to the obstacle. This speed dynamics can chosen as follows:

$$V_k(t) = \begin{cases} (V_{leader}(t) + p|\delta_{speed}|)V_O(t), & \varphi_F > 0 \\ 0 & \varphi_F \leq 0 \end{cases} \tag{7.40}$$

where p is a positive constant parameter used to adjust the rate of correction, and $\delta_{speed} \in [-1, +1]$ is the correction term. Its value varie depending on how far a follower from its desired position is.

formation-steer

According to figure (7.11), desired positions for the configurations *line* and *traingle* are:

$$\begin{cases} x_2 = x_{leader} + dist_2 \sin(\theta_{leader} - \alpha_2) \\ y_2 = y_{leader} - dist_2 \cos(\theta_{leader} - \alpha_2) \end{cases} \tag{7.41}$$

$$\begin{cases} x_3 = x_{leader} - dist_3 \sin(\theta_{leader} + \alpha_3) \\ y_3 = y_{leader} + dist_3 \cos(\theta_{leader} + \alpha_3) \end{cases} \tag{7.42}$$

where $dist_2$ and $dist_3$ are the distances of the followers 2 and 3 relative to the leader, respectively. In the case of *line* formation, $\alpha_2 = 0$ and $\alpha_3 = 0$.

For the *column* formation (Figure 7.12) , desired positions are:

$$\begin{cases} x_2 = x_{leader} - dist_2 \cos(\theta_{leader}) \\ y_2 = y_{leader} - dist_2 \sin(\theta_{leader}) \end{cases} \tag{7.43}$$

$$\begin{cases} x_3 = x_{leader} - dist_3 \cos(\theta_{leader}) \\ y_3 = y_{leader} - dist_3 \sin(\theta_{leader}) \end{cases} \tag{7.44}$$

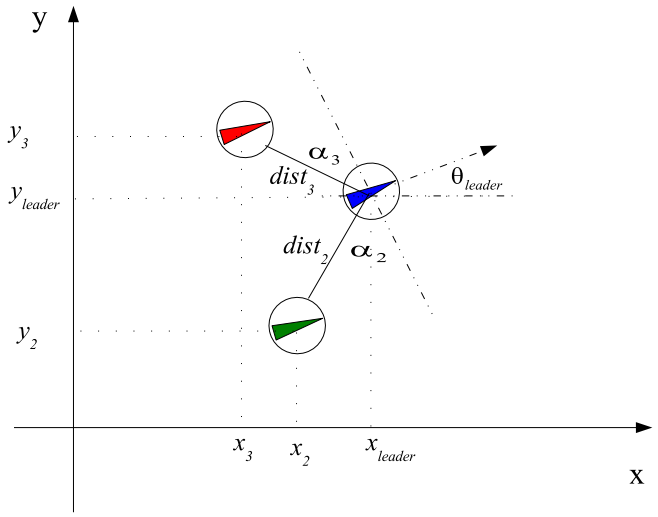


Figure 7.11: Triangle and Line Configuration

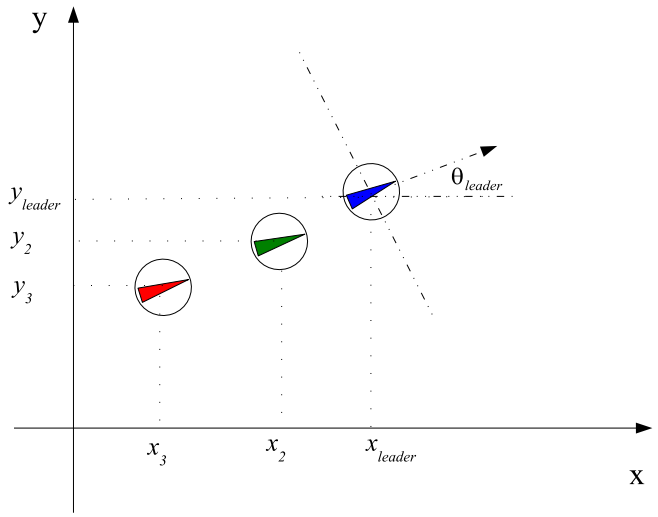
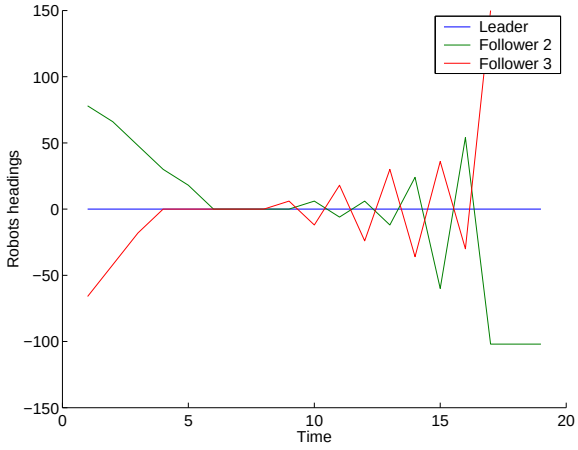


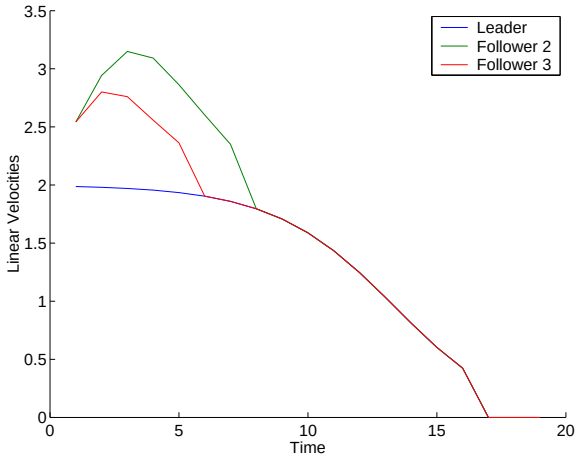
Figure 7.12: Column Configuration

7.7 Results

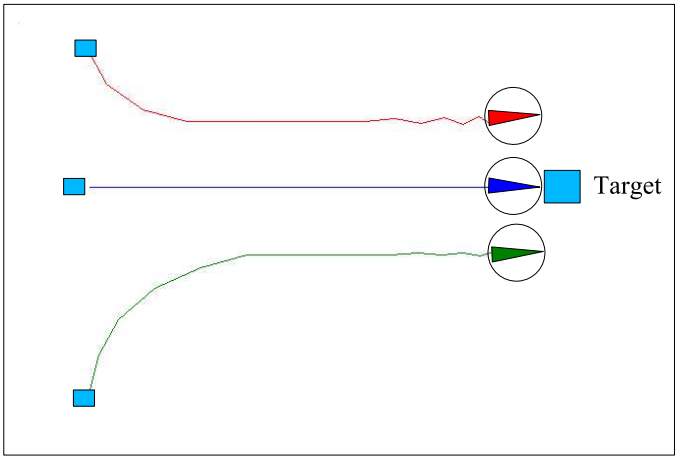
In order to update the field stimulus, we assume that each robot has a 360^o sensor. The sensor range is assumed to be a circle with a radius equal to 1. On each neural field we chose $N = 60$ neurons, which means that each direction N decodes a step of 6^o. Figure (7.13) shows a simulation run, where the objective is to maintain the *line-abreast* formation. At the beginning the followers started with a higher speed, in order to reach the desired position, since they started away from the leader (Figure 7.13 a). Once the required positions are reached, they move with the same heading and speed of the leader. The next simulations evaluate the *obstacle-avoidance* and *maintain-formation* performance. Figures (7.14) and (7.15) show the obtained results for avoiding obstacles and acquiring static and moving targets. When a robot is close to an obstacle, the stimulus contain obstacle entries, which brought the robot away from the obstacle. This behavior was expected, since *obstacle-avoidance* behavior has the highest priority. The same behaviors can observed on figures (7.16), (7.17) and (7.18) for the formation *triangle*. The result of *colmun* formation are presented in Figure(7.19). *colmun* formation is also tested in the case of door passing (Figure 7.20). In this test, the control approach has succefully provided the robots the ability to traverse through a narrow gap, and maintaining the *colmun* formation.



(a)

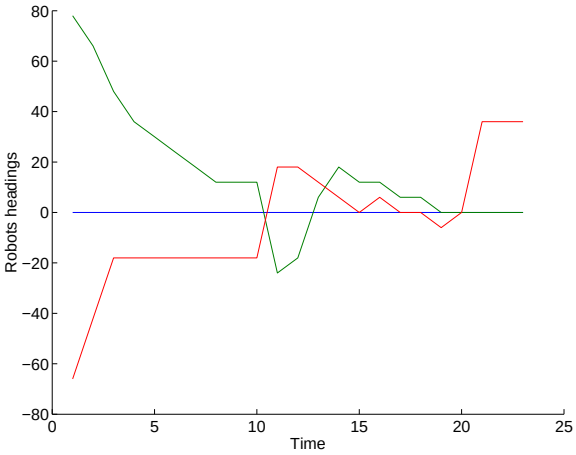


(b)

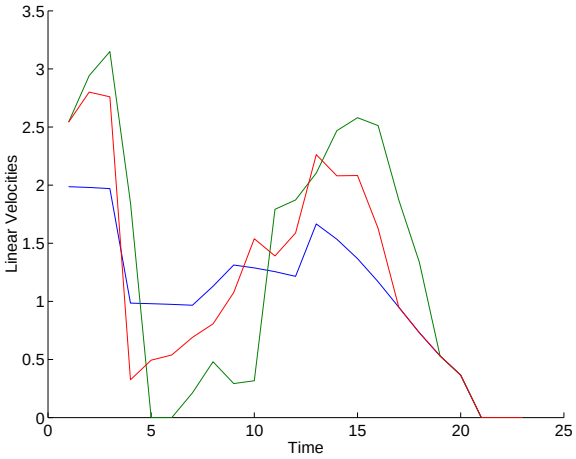


(c)

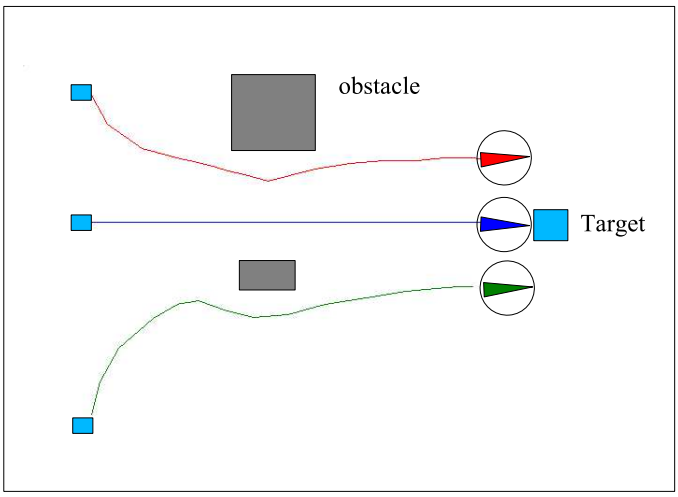
Figure 7.13: *Line* configuration in obstacle free situation. a)Robots' headings. b) Robots' velocities



(a)

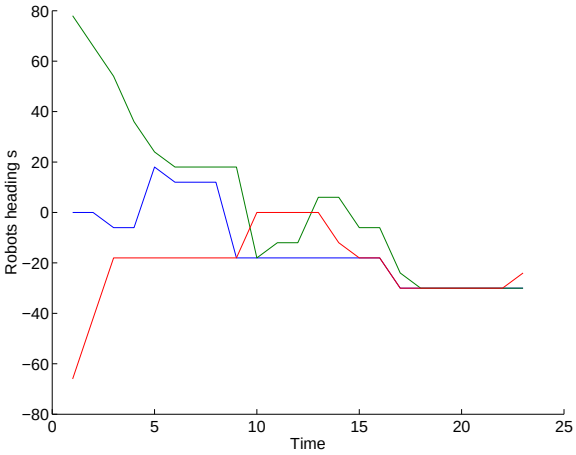


(b)

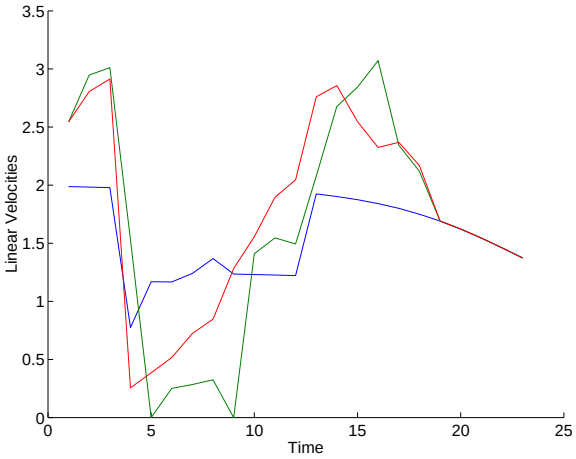


(c)

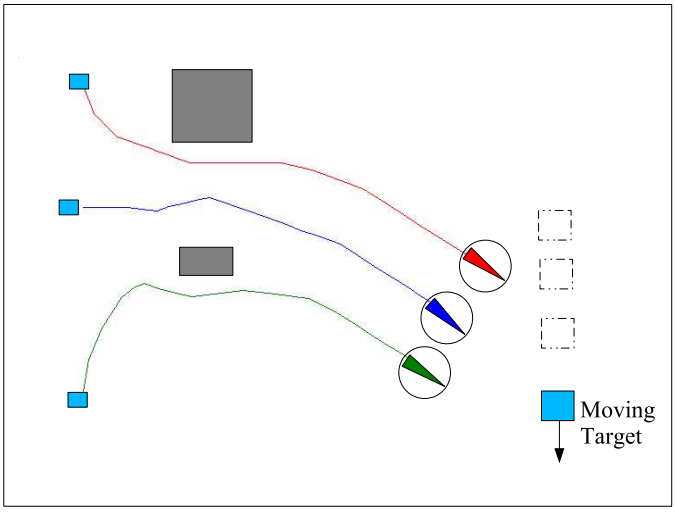
Figure 7.14: *Line* configuration with obstacles. a)Robots' headings. b) Robots' velocities



(a)

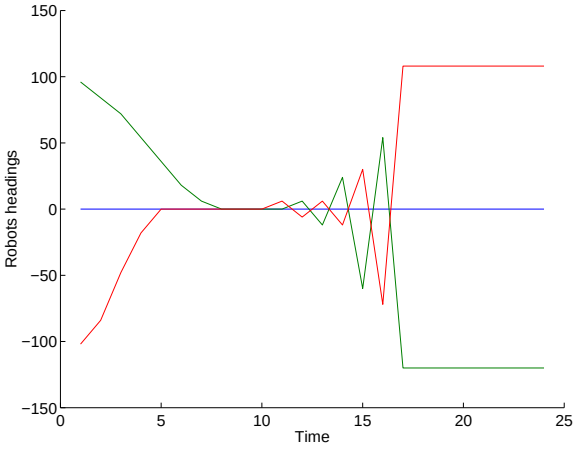


(b)

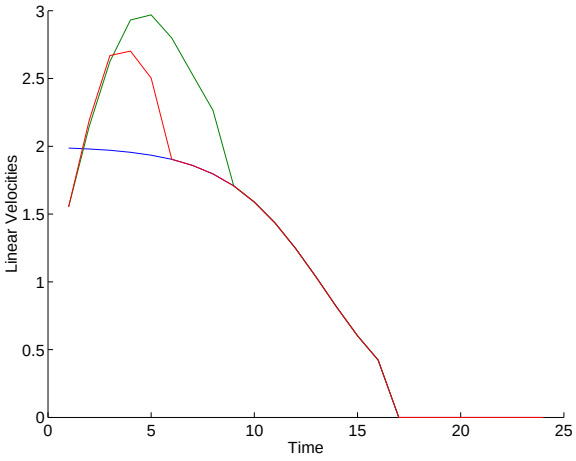


(c)

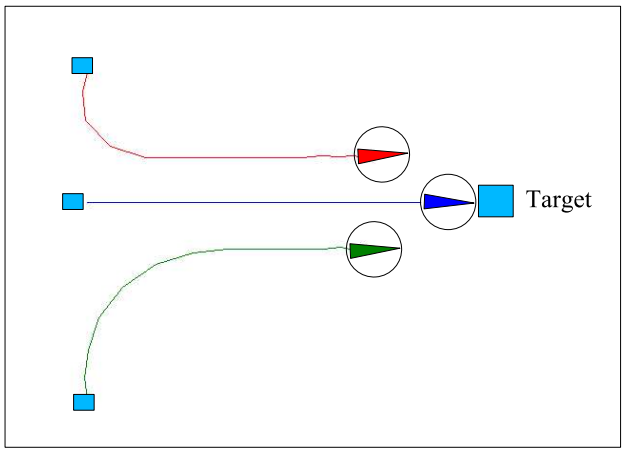
Figure 7.15: *Line* configuration with moving target and fixed obstacles. a)Robots' headings. b) Robots' velocities



(a)

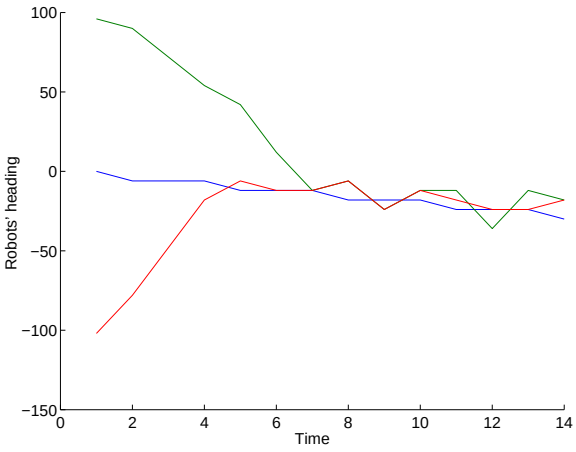


(b)

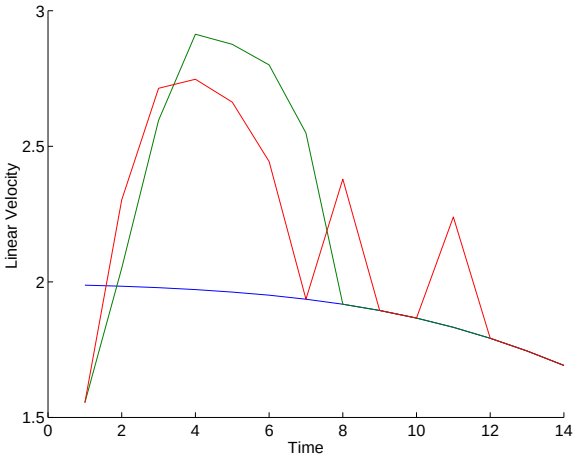


(c)

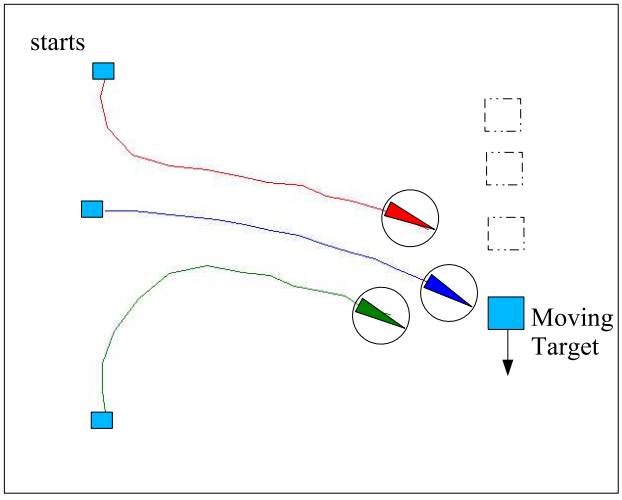
Figure 7.16: *Triangle* configuration. a)Robots' headings. b) Robots' velocities



(a)

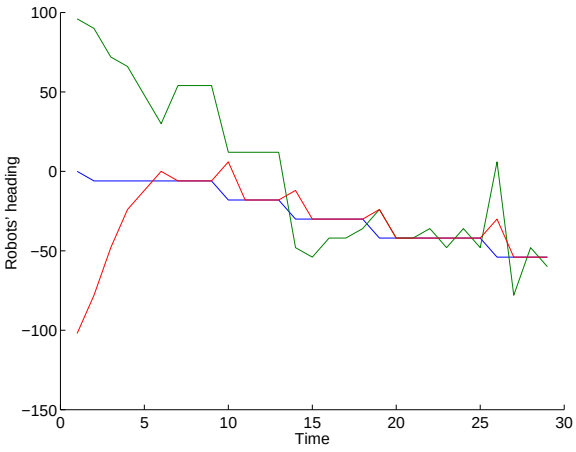


(b)

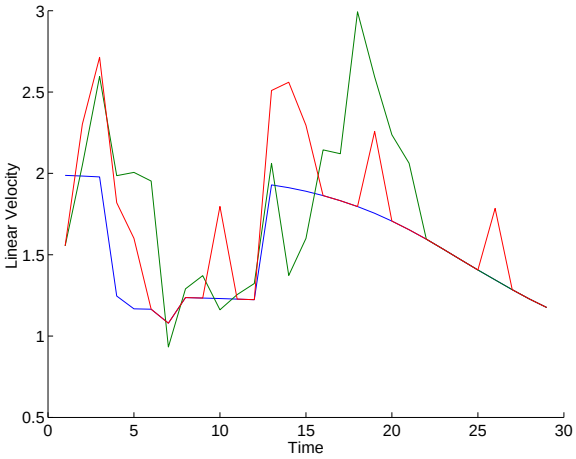


(c)

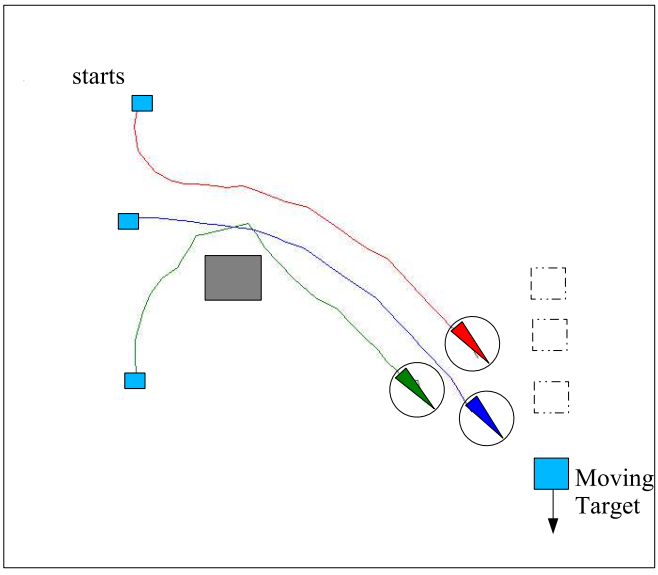
Figure 7.17: *Triangle* configuration with moving target. a)Robots' headings. b) Robots' velocities



(a)

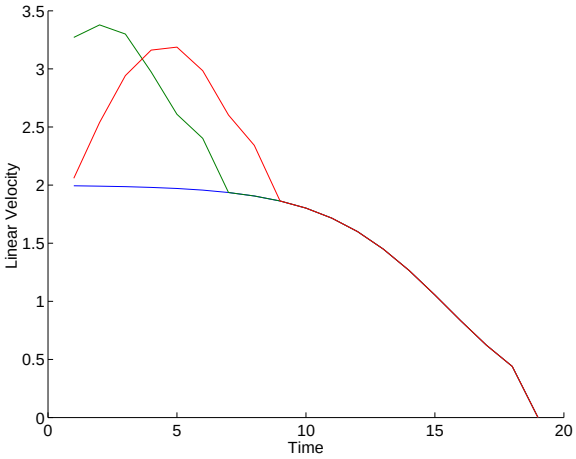


(b)

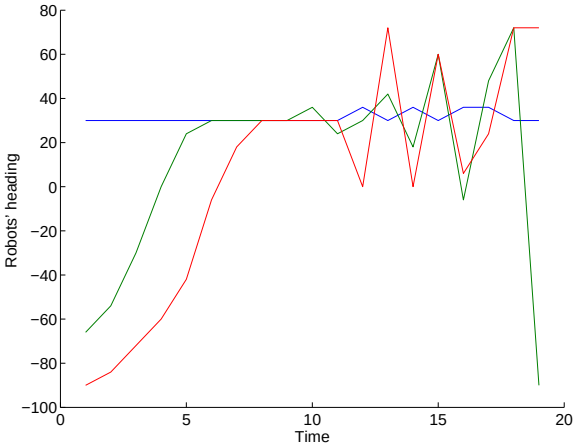


(c)

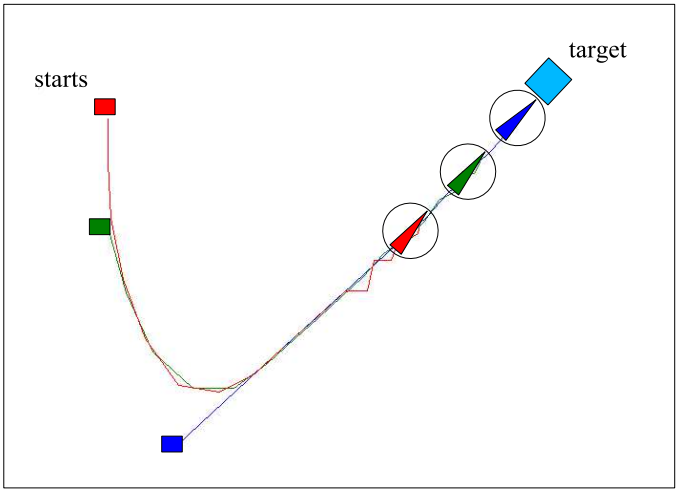
Figure 7.18: *Triangle* configuration with moving target and fixed obstacles. a)Robots' headings. b) Robots' velocities



(a)

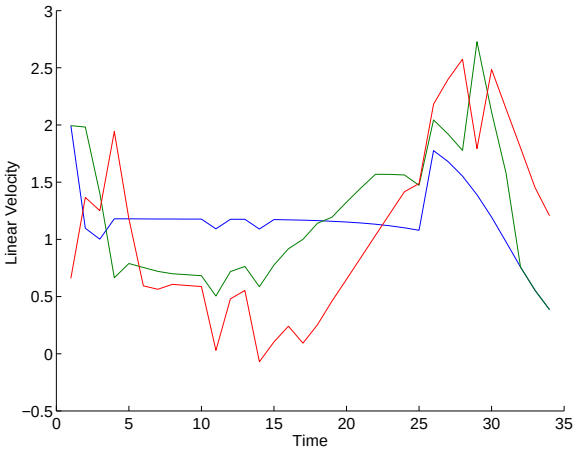


(b)

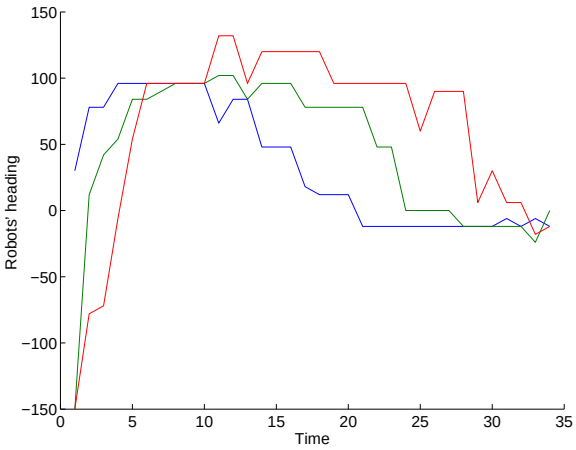


(c)

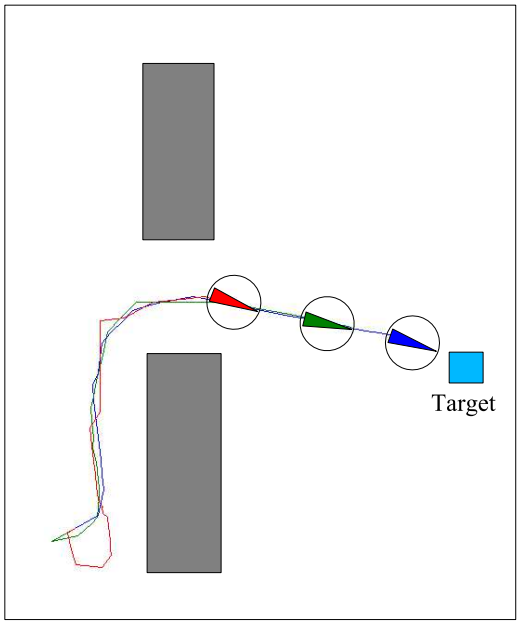
Figure 7.19: *Column* configuration in a door passing test. a)Robots' headings. b) Robots' velocities



(a)



(b)



(c)

Figure 7.20: *Column* configuration in a door passing test. a)Robots' headings. b) Robots' velocities

7.8 Conclusion

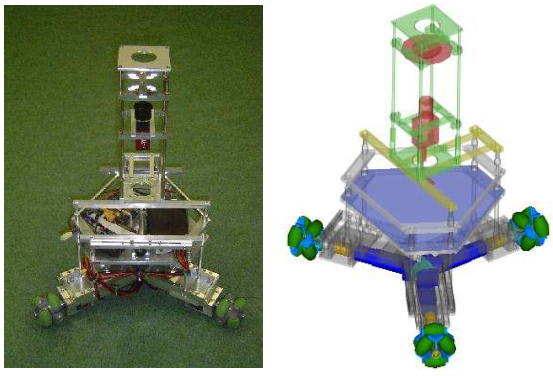
In this chapter, neural fields have been chosen as a framework for the behavior based robot control. While their suitability has been already proven to solve the problem of target-acquisition with obstacle avoidance [110][111], we described here how neural fields could be used in more complex problems: avoiding static and moving obstacles, multi-target acquisition, and maintaining formation for multiple robots. The navigation model is developed in order to produce peak-solutions of the field activation, which encode the appropriate robot direction in response to a change in the environment. Through competitive dynamics, the *sub-target neural field* expresses, whether a home-base is the next sub-target or not. A Peak will decode the direction to a sub-target whenever one, or no peak otherwise. Following this method, the entire optimal trajectory from the start to the main target was not necessary to be known, but only local information about the next sub-target. This implies that the optimal decisions were not static: a change in the accessibility of a home-base leaded to change the decision on which optimal home-base to visit next. For multirobot control, we have decomposed the problem of formation control into: 1) control of a single lead robot and 2) control of two follower robots in the team. The lead robot has to acquire the main target, while the follower robots maintain formation. In addition, all robots have to avoid obstacles and collision with each other. Based on these requirements, the three neural fields were provided by the necessary stimulus information. Three geometric configurations were considered: *line*, *column*, and *triangle*. For each configuration, two tasks, *formation-speed* and *formation-steer* run simultaneously to keep each robot in its desired position. Simulations results demonstrate the smooth behavior of the robots, despite all imposed constraints. Figure (7.18) demonstrates that tracking a moving target, avoiding obstacles, and maintaining triangle formation is not an absolute limitation of our approach. The scenario of passing a door in a column formation did not present any difficulty as well (figure (7.20)). Furthermore, in all developments, the speed control is fully integrated here. In earlier studies, this quantity was usually set to a constant value.

All in all, simulations results demonstrate that the neural field concept could provide an elegant solution for behavior-based control of single or multiple mobile robots, despite many imposed constraints. A real implementation of the control design will be presented in chapter (8).

Chapter 8

Neural Fields for Behavior-Based Control of a RoboCup Player

As seen in previous simulation results, neural fields provide an elegant local solution for behavior control. Due to their low-computational costs, this approach is suitable for real application where the environmental state changes continually. This chapter presents a set of experiments in a real environment. The main objective is to move a robot to a target while avoiding static and moving obstacles. Experimental tests were obtained using the same robot (figure 8.1) used in chapter (6).



(a) (b)

Figure 8.1: Omnidirectional robot. a)hardware photo. b) CAD model

8.1 Robot System

The neural field approach is implemented on one robot of the CoPS-Team (Cooperative Soccer Playing Robots) of Stuttgart University [116]. It is equipped with 3 omni-wheels, each of them driven by a 90W DC motor. Gearboxes with reduction ratios of 14:1 are used to reduce the high angular speeds of the motors (7000 rpm) and to amplify the wheel's mechanical torques, and 500 ppr digital incremental encoders are used to measure the actual wheels speed. Motors are controlled by 3-channel microprocessor-based interface. The robot is equipped with a laptop to manage different sensors and tasks. The communication between the sensors and the laptop can be done through USB, RS232, or IEEE1394 (FireWire). A pneumatic kicker supplied by a small air pressure tank is also installed on the robot.

8.1.1 Environment Sensing

For environment sensing, the robot platform is equipped with an omnidirectional vision system, based on a hyperbolic mirror and a standard IEEE1394 (FireWire) camera. The Omnidirectional vision provides the robot a very large field of view, which has some useful properties. For instance, it can facilitate the tracking of robots, the ball, and a set of environmental features used for self-localization. To extract information, the captured image is segmented using the calibrated colors of relevant objects, such the ball, field lines, and obstacles. An example of an image with recognized objects is shown in Figure (8.2).

(a) (b)

Figure 8.2: Information extraction from the camera a) Image captured from the omni-camera b) Recognition of relevant objects: lines(white), ball(red), obstacles (black), and goals (blue and yellow).

8.1.2 Self-Localization

The problem of self-localization is to estimate the robots pose relative to its environment. In our robots we use a probabilistic localization algorithm called Monte Carlo Localization (MCL) [117]. MCL can solve the localization problem in a highly robust and efficient way, even with temporary partial or total occlusion of relevant sensor features.

8.1.3 Software Architecture

The Software architecture has a modular design structured into three parallel working layers (Figure 8.3):

1 *Sensor layer*, where low level features or raw data are gathered,

2 *World Model layer*, which is used for storage and administration of data objects. It provides so-called Data Processor modules. These modules work on dedicated data objects of the world model to generate data of a higher degree of abstraction. Monte Carlo based localization and Kalman filter based ball tracking are examples of theses modules. The

world model stores locally gathered data as well as communicated data by other robots.

3 *Control layer*, responsible for the execution of actions. While its Pilot module simply provides driving and steering capabilities, its Navigator module implements complete soccer behaviors like driving or dribbling to a position or shooting a ball.

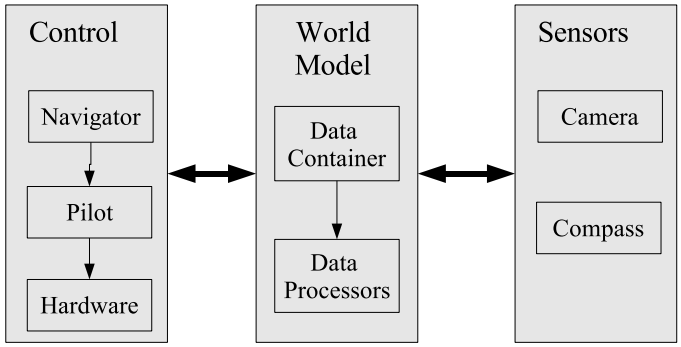


Figure 8.3: Robot Software Architecture.

8.2 Neural Fields

For convinient, we rewrite the field equation of a one-dimensional neural field:

$$\tau \dot{u}(\varphi, t) = -u(\varphi, t) + S(\varphi, t) + h + \int_{-\infty}^{+\infty} w(\varphi, \acute{\varphi}) f(u(\acute{\varphi}, t) d\acute{\varphi} \tag{8.1}$$

where $u(\varphi, t)$ is the field excitation at time t ($t \geq 0$) at the position $\varphi \in R$. The temporal derivative of the excitation is defined by

$$\dot{u}(\varphi, t) = \frac{\partial u(\varphi, t)}{\partial t} \tag{8.2}$$

The excitation $u(\varphi, t)$ of the field varies with the time constant τ with $\tau \in R^+$. The constant h defines the pre-activation of the field, and $f(u)$ is the local activation function. Usually, f is chosen as a step-function:

$$f(u) = \begin{cases} 1, & u \geq 0 \\ 0 & u < 0 \end{cases} \tag{8.3}$$

The stimulus $S(\varphi, t) \in R$ represents the input of the field which is dependent on the field position and varies with time. A nonlinear interaction between the excitation $u(\varphi)$ at the position φ and its neighboring positions is achieved by the convolution of an interaction kernel $w(\varphi, \acute{\varphi})$.

To integrate neural fields in the software system of the robot, equation (8.1) is discretised as:

$$\frac{\tau}{\Delta t}[u(i, t+\Delta t) - u(i,t)] = -u(i,t) + \sum_{j=-\infty}^{+\infty} w(i,j) f[u(j,t)] + S(i,t) + h \quad (8.4)$$

where Δt is the discretised time, and $w(i,j)$ is the interaction kernel between neurons i and j. $S(i,t)$ is the external stimulus at the neuron i. To specify the dynamic properties of the field, the term $W(x)$ (see chapter 7) will be defined as:

$$W(x) = \sum_{j=0}^{x} w(j) \quad (8.5)$$

8.2.1 Equilibrium Solutions

In the absence of an external stimulus ($S(i,t) = 0, \forall j$), the equilibrium solutions satisfy

$$u(i, t+\Delta t) = u(i,t) : \forall i \quad (8.6)$$

The field equation will be independant of time

$$u(i) = \sum_{j=-\infty}^{+\infty} w(i,j) f[u(j)] + h \quad (8.7)$$

As in the continuous case, the model (8.4) has three equilibrium solutions, according to the theorems (1) and (2) in chapter (7):

1. There exists a $\emptyset$-solution if and only if $h < 0$.

2. There exists an ∞-solution if and only if $2W_{\infty} > -h$.

3. There exists an a-solution (a local excitation of length a) if and only if $h < 0$ and $a > 0$ satisfies $W(a) + h = 0$.

4. The a-solution is asymptotic stable if $\frac{dW(a)}{da} < 0$, and unstable if $\frac{dW(a)}{da} > 0$.

As seen in chapter (7), if the field is in a-solution and an input of a stimulus $S(i,t)$ at time t is very large compared with the within-field cooperative interaction, this input will dominate the solution. As a result, a single-peak will be stabilized by interaction, even if the stimulus is removed.

8.3 Control Design

In the experiments, the neural field has to encode angles from $-\pi$ to $+\pi$. By means of a codebook we use N discrete directions. The one-dimensional neural field for the robot directions will be:

$$\tau u(i,t+\Delta t) = (\tau - \Delta t)u(i,t) + \Delta t \left[\sum_{j=1}^{N} w(i,j) f[u(j,t)] + S(i,t) + h \right] \quad (8.8)$$

The neural parameters field are chosen as follows. The pre-activation h of the filed is fixed to $h = -1$, the time constant to $\tau = 2$, and the discret time $\Delta t = 0,2$. The interaction kernel is chosen as:

$$w(i,j) = 5e^{-0.5(i-j)^2} - 2 \quad (8.9)$$

8.3.1 Field Stimulus

Before selecting an appropriate robot direction, the neural field needs some necessary information (stimulus). The stimulus is determined according to two stimulus-functions. These functions describe

1 the direction towards the target $S_T(i,t)$. This stimulus is designed excitatory, showing an attraction towards the target direction. It is chosen as

$$S_T(i,t) = C_{T1} - C_{T2}|i - i_T(t)| \quad (8.10)$$

where $C_{T1} = 25$, $C_{T2} = 1$, and $i_T(t)$ is the field position, equivalent to the target direction at time t.

2 directions to obstacles $\{S_{Ol}(i,t) : l \in [1,N]\}$, where N_{Obst} are the number of obstacles detected by the robot sensors. However, the stimulus consider only obstacles, which their distances d_{Ol} to the robot are less than

a threshold $d_{Th} = 1m$. This stimulus must be inhibitory, since obstacles collision must be avoided. It is chosen as a Mexican Hat function centered at the direction of an obstacle.

$$S_{Ol}(i,t) = C_O e^{-\sigma_O(i-i_l)^2} \tag{8.11}$$

where $C_O = 50$,and $\sigma_O = 0.05$ which defines the range of inhibition of an obstacle.

The contribution of different stimulus defines the state of the field. Thus, the stimulus of the field at time t is determined by

$$S(i,t) = S_T(i,t) - \sum_{l=1}^{N_{Obst}} g(d_{Ol}) S_{Ol}(i,t) \tag{8.12}$$

where g is a step function:

$$g = \begin{cases} 1, & d_{Ol} < d_{Th} \\ 0 & d_{Ol} \geq d_{Th} \end{cases} \tag{8.13}$$

8.3.2 Dynamics of Speed

In a free obstacle situation, the robot can move with its maximum speed $V_{\max} = 1.5m/s$. However, close to obstacles or a target the robot needs to be slowed down. In free obstacles, the velocity can be chosen as:

$$V_T(t) = V_{\max}(1 - e^{-\sigma_v d_T}) \tag{8.14}$$

where $\sigma_v = 0.8$, and d_T represents the distance of the robot relative to the target at time t. When the robot approaches obstacles, its velocity must be reduced relative to the obstacle distance and its direction. In case of many obstacles, the nearest obstacle direction to the robot movement direction is considered. This dynamics can be chosen as:

$$V_O(t) = C_O(1 - g(d_{no}) e^{-\sigma_{Ov}(\varphi_F - i_{nO})^2}) \tag{8.15}$$

where $C_O = 1$ and $\sigma_{Ov} = 0.003$. i_{nO} is the nearest obstacle direction to the robot movement direction and d_{nO} is its distance relative to the robot. The final dynamics of the velocity is the contribution of (8.14) and (8.15). Furthermore, it is also considered when no appropriate direction can be selected, for example when the robot is completely surrounded by obstacles.

In this case the robot must stop until the environmental situation changes. Thus, the robot velocity that satisfies the above design creteria is the following:

$$V(t) = \begin{cases} V_T(t)V_O(t), & \varphi_F > 0 \\ 0 & \varphi_F \leq 0 \end{cases} \tag{8.16}$$

8.4 Results

Expriments were made on a soccer field of $10,421m$ length and $5,720m$ width. The origin of the coordinates is taken on the middle of the field. On the neural field we chose $N = 60$ neurons, which means that each direction N decodes a step of 6^o. In all experiments, the initial and the target positions of the robot are given in the spatial coordinates and equal to $(-3000, 0)$, $(3000, 0)$, respectively. The neural field is expected to align the robot's heading with the direction of the target, and to bring it away from the nearby obstacles. We will illustrate this with some experiments showing target acquisition with collision avoidance for multiple static and moving obstacles.
Figure (8.4) shows the results of the first experiment. In the first 20 time steps, both, the stimulus (Figure 8.4.b) as well as the field activation (Figure 8.4.a) are unimodal, since no obstacles are detected and the stimulus contains only the target entries. In the following time steps, the stimulus contains also obstacle entries. Therefore, it becomes bimodal. By contrast, on the field activation, the peak follows the optimum direction produced by the *target-acquisition* and *obstacle-avoidance* behaviors. It provides now the appropriate heading direction towards the target with obstacle avoidance (Figure 8.4.d). We can see also how the robot speed is slowed down when it is near the obstacle or approaches the target (Figure 8.4.c). In this experiment, the distance between the obstacles 2 and 3 is not enough to contain the robot dimension. Due to the contribution of the two obstacle stimulus the robot was obliged to pass to the right and avoid obstacle 3 in order to reach the target. Figure (8.4.e) shows the entire path movement of the robot from its initial position to the target position. Photos from video sequences are presented in figure (8.6). In the second experiment (figure 8.5), the distance between the obstacles 2 and 3 is now large enough to contain the robot dimension. The robot could move through these two obstacles to reach the target. Figure (8.7) shows some photos from video sequences of the second experiment.
The third experiment (figure 8.8) was carried out to verify the navigation

approach ability in tackling moving obstacles. In this context, the problem is how to reach a target in the presence of dynamically moving obstacles. During the first 20 time steps, both, the stimulus (figure 8.8.b) as well as the field activation (figure 8.8.a) are unimodal, since the stimulus contains only the target entries. In the following time steps, the robot avoids the obstacle 1. At time step 51, the stimulus contains obstacle 2 entries. In the next 15 time steps, the neural field provides the appropriate heading direction towards the target with obstacle 2 avoidance (Figure 8.8.d). After passing obstacle 2, the entries of the target became stronger. The peak changed its position towards the direction of the target. Figure (8.8.e) shows the whole path movement of the robot.

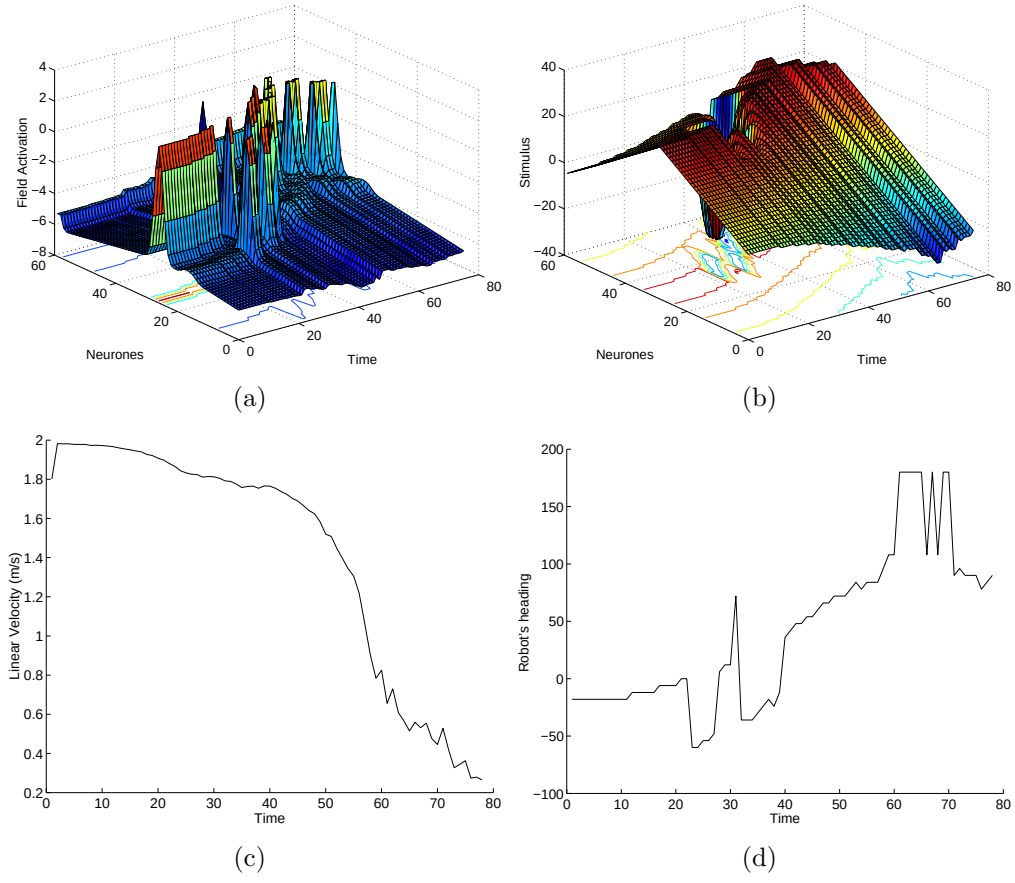
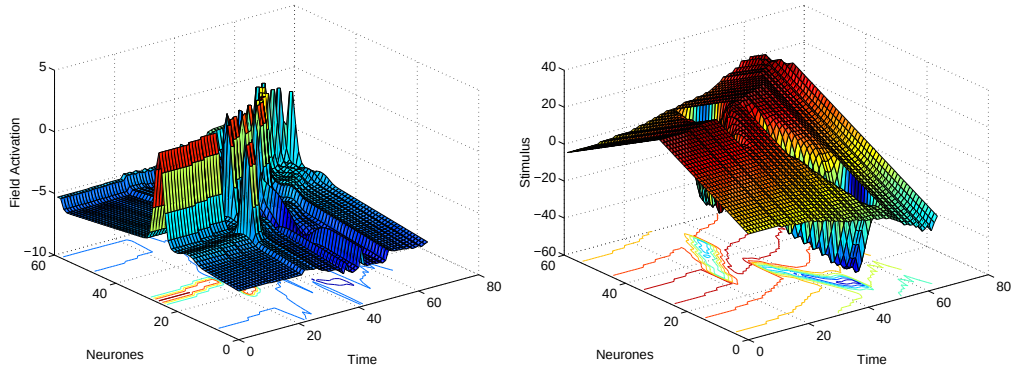
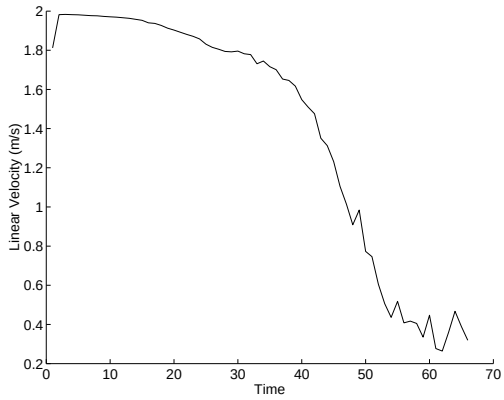


Figure 8.4: Target acquisition with Obstacle avoidance: first experiment

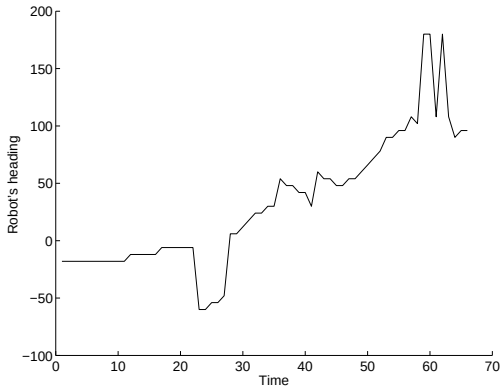


(a)

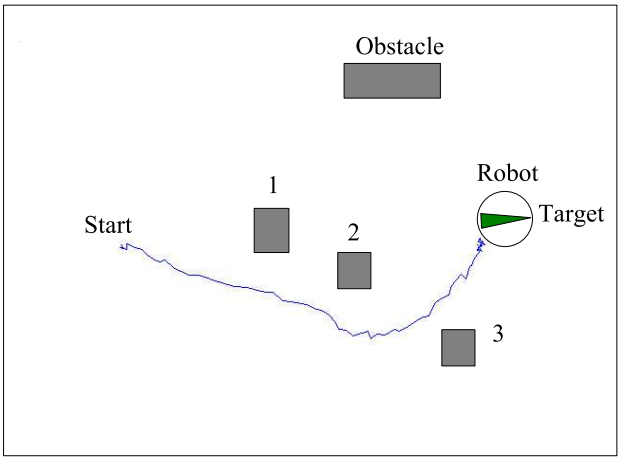
(b)



(c)



(d)



(e)

Figure 8.5: Target acquisition with Obstacle avoidance: second experiment

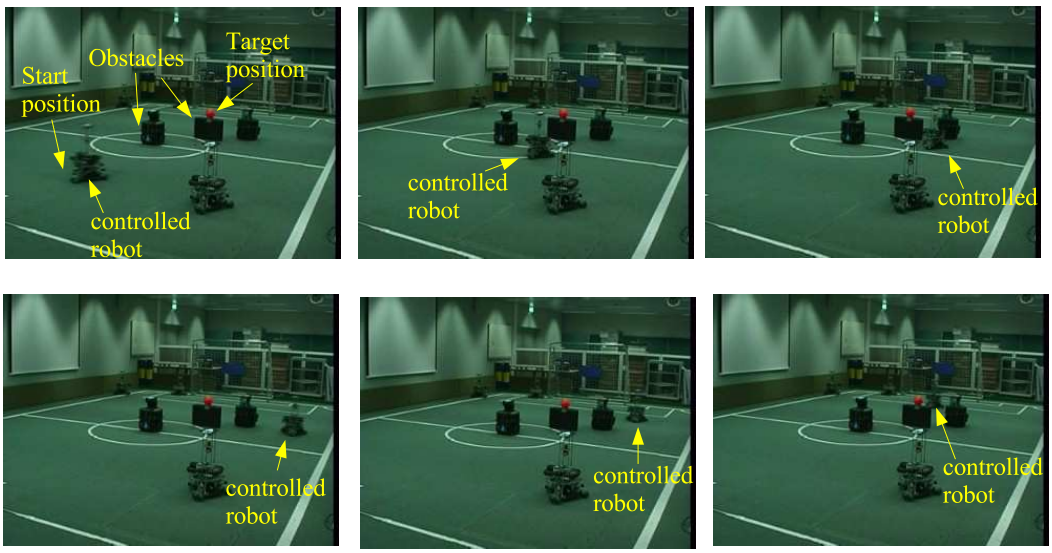


Figure 8.6: Photos from video sequences of the first experience.

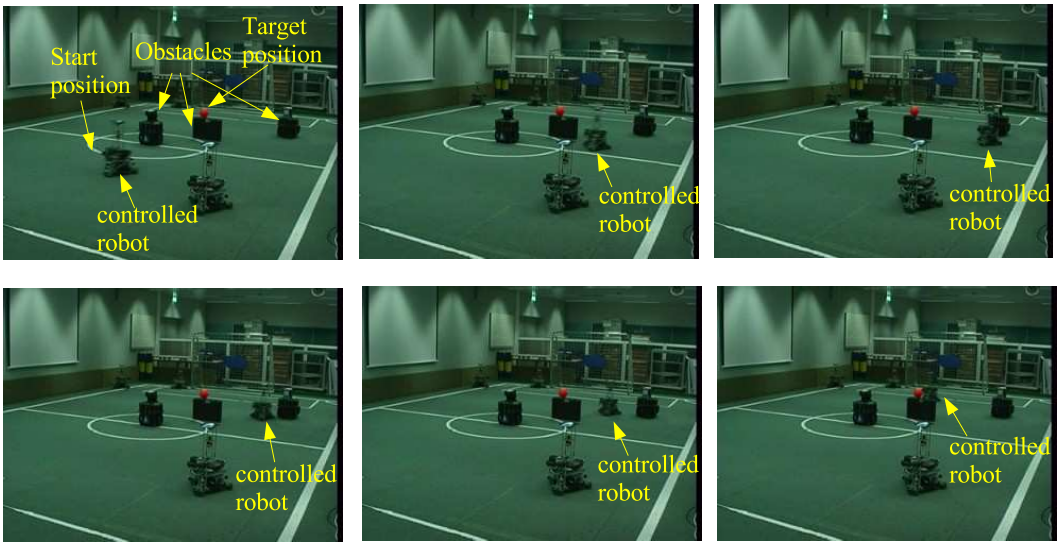


Figure 8.7: Photos from video sequences of the second experience.

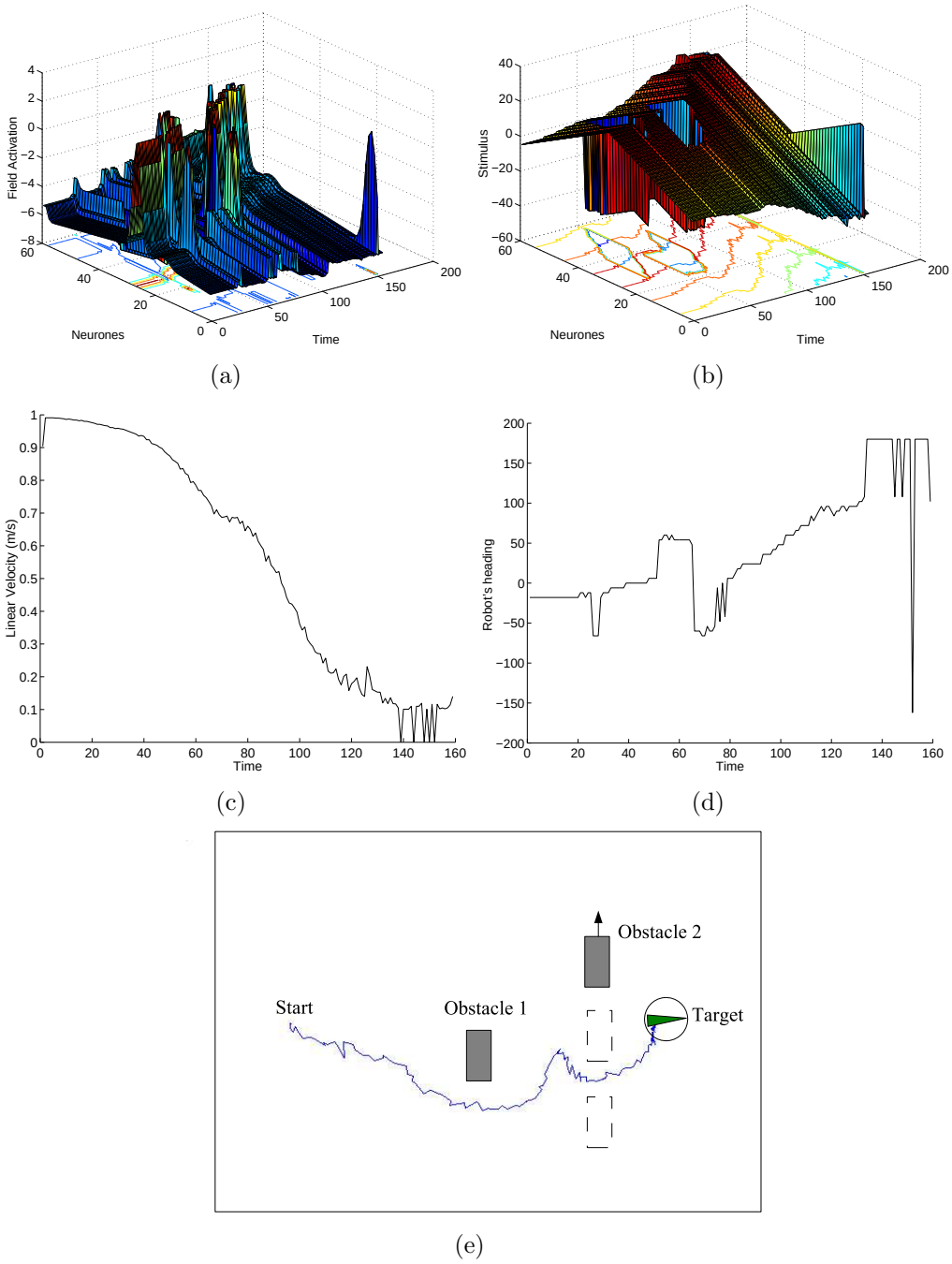


Figure 8.8: Target acquisition with moving obstacle avoidance

8.5 Conclusion

In this chapter, we have demonstrated how neural fields have been chosen as a framework for the behavior-based control of a RoboCup Player. The navigation model is developed in order to produce peak-solutions of the field activation, which encode the appropriate robot direction in response to a change in the environment. The robot had to acquire the main target, while avoiding static and moving obstacles.

This chapter demonstrated the utility of the approach only on one robot. One issue that needs to be addressed in future research is to implement the approach on multiple robots. In RoboCup, formation control plays a major role. For example, if a lead robot loss the ball, the other follower robots could catch it and continue the movement. Another application is in the defense position, where the robots should form a "wall" around the goal area.

Chapter 9

Conclusions and future work

9.1 Summary of Contributions

When we started this work, we had two goals: explore recurrent neural networks, and develop solutions for two robotics problems.

We studied the idea that a large recurrent network can serve as a source of dynamics from which information can be extracted by a readout unit. It was adopted as a solution to overcome some RNNs training difficulties. Also, we explored the concept of metalearning to train a RNN that can adapt without changing explicit weights.

Regarding robotics, we firstly designed a motion control approach based on a novel random recurrent neural network called echo state network (ESN). The advantage is that no knowledge about the robot model is required, and no synaptic weight changing is needed in presence of parameters variation. In addition, this ability allowed a single fixed weight ESN to act as a dynamic controller for several distinct mobile robots. In behavior-based control, the concept of neural fields theory was adopted to generate the robot behavior over time. We described how neural fields could be used to move a robot towards a target while avoiding static or moving obstacles. The naviga-

tion model was developed in order to produce peak-solutions of the field activation, which encode the appropriate robot direction in response to environment changes. Through their competitive dynamics, we also designed a framework to optimize the target path through intermediate home-bases. Moreover, we have demonstrated how neural fields are able to generate behavior control for multiple mobile robots. The objective was to move a team of robots to a target while avoiding obstacles and keeping a geometric configuration.

9.2 Conclusions

In this thesis we investigated how dynamics in recurrent neural networks can be used to solve some specific mobile robot problems. The ability of RNNs to handle arbitrary temporal dynamics, and robustness for noise environment make them an ideal choice for real dynamical systems control. However, this potential is not being exploited because simple and powerful training algorithms were missing. To overcome this problem we adopted the concept of echo state networks (ESNs). ESN has an easy training algorithm where only the network-to output connection weights have to be trained using a linear regression. The training experiments carried out here demonstrate that small and partially interconnected ESNs could be trained to act as motion controllers for mobile robots. Moreover, ESNs were being asked to deal with parameters variation. In other words, they were asked to exhibit a characteristic, normally ascribed to adaptive controllers, whose parameters change in response to an environmental change. "Adaptation" in this work is defined as the ability of an ESN to recognize change only through the system output and its own state, without changing any synaptic weight. This capability is a natural consequence of prior metalearning used during training. When a change occurred, the state of ESN switched from one family of orbits to another, which corresponds to the change. This ability allowed us also to design a single fixed-weight controller for multiple distinct mobile robots.
The fixed-weight solution adopted here presents a clever and efficient strategy compared with adaptive networks. Furthermore, no multi-streaming is needed during training, since ESNs use a batch-learning algorithm and do not suffer from the recency effect.

Neural fields have been chosen as a framework for the behavior-based

control. While their suitability has been already proven to solve the problem of target-acquisition with obstacle avoidance, we described here how they could be used in more complex problems: acquiring one or multiple targets, avoiding static and moving obstacles, and maintaining formation for multiple robots. The navigation models were developed in order to produce peak-solutions of the field activation, which encode the appropriate robot direction in response to a change in the environment. The entire optimal trajectory from the start to the main target was not necessary to be known, but only local information was needed. This implies that the optimal decisions were not static: a change in the accessibility of a home-base for example leaded to change the decision on which optimal home-base to visit next.
In multi-robot control, two behaviors, *formation-speed* and *formation-steer* run simultaneously to keep each robot in its desired position.
The obtained results demonstrate the smooth behaviors, despite all imposed constraints.

All in all, we have shown that dynamics in RNNs could provide simple and elegant solutions for some problems in mobile robotics. Successful results have been obtained on simulations and on real implementations. However, several open issues still remain for future research.

9.3 Future Directions

Although we consider that the objectives of this thesis have been accomplished, there are plenty of improvements that could be done in order to achieve better results. Here, only the main points are summarized.

- We are aware of a certain degree of arbitrariness in our choice of the ESN parameters and architecture. Substantial investigation and more experiments on much larger data sets are still needed to ensure that the results we have achieved to date are indeed statistically significant.

- How to guaranty stability? As any RNN, a relatively large dimension ESN lost stability at many times and exhibited sometimes high-frequency oscillations on smooth test signals.

- What is the behavioral capacity of RNNs? In this work, we explored the performance of ESNs to exhibit adaptive behavior with fixed-weights.

However, we did not discuss the question whether the obtained fixed-weights ESNs could pass any test in adaptive system. Our network works well only for those situations, for which it has been trained. This may be seen as a limitation relative to explicitly adaptive systems. Does a more efficient training method exist?

- We have provided a control framework based on neural fields theory. Much more efforts must be done to completely integrate movement planning and motor control. One important issue is to integrate the speed control with the temporal evolution of neural activation. A first step towards addressing this issue is to find out what are the optimal neurons activation states before and after performing movements. Further step is to estimate the optimal timing of the movement.

- Only simulation results demonstrated the neural fields utility for multi-robot system. This approach must be implemented and tested on a team of real robots (for example a team of RoboCup robots).

Bibliography

[1] Mataric Maja J. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents*, 9:323–336, 1997.

[2] Erann Gat. On three-layer architectures. *Artificial Intelligence and Mobile Robots,MIT/AAAI Press*, 1997.

[3] Connell Jonathan. A hybrid architecture applied to robot navigation. In *Proc. of the 1992 IEEE International Conf. on Robotics and Automation*, pages 2719–2724, 1992.

[4] Arkin R. Towards the unification of navigational planning and reactive control. In *American Association for Artificial Intelligence Spring Symposium on Robot Navigation*, pages 1–5. AAAI/MIT Press, 1989.

[5] Brooks Rodney A. A robot that walks; emergent behavior from a carefully evolved network. *Neural Computation*, pages 253–262, 1989.

[6] Connell Jonathan. Designing behavior-based robots. In *SPIE-91 Conference on Mobile Robots*, volume 1613, pages 34–45, 1991.

[7] Brooks Rodney A. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, Mar 1986.

[8] Brooks Rodney A. New approaches to robotics. *Science*, 253:1227–1232, 1991.

[9] Mataric Maja J. Navigating with a rat brain: a neurobiologically-inspired model for robot spatial representation. In *First International Conference on Simulation of Adaptive Behavior*, pages 169–175, Cambridge,MA, 1990. MIT Press.

[10] Mataric Maja J. Integration of representation into goal-driven behavior-based robots. *IEEE Transactions on Robotics and Automation*, 8(3):304–312, 1992.

[11] Pirjanian P. *Multiple Objective Action Selection and Behaviour Fusion using Voting*. PhD thesis, Aalborg University, 1998.

[12] Jana Kosecka and Ruzena Bajcsy. Discret event systems for autonomous mobile agents. In *Proceedings of Intelligent Robotic Systems*, pages 21–31, July 1993.

[13] Long-Ji Lin. Scaling up reinforcment learning for robot control. In *Proceedings of the 10th International Conference on Machine Learning*, 1993.

[14] Steen Kristensen. *Sensors Planning with Bayesian Decision Analysis*. PhD thesis, Department of Medical Informatics and Image Analysis, Aalborg University, 1996.

[15] Pirjanian P. Behavior coordination mechanisms – state-of-the-art. Technical Report IRIS–99–375, Institute of Robotics and Intelligent Systems, School of Engineering, University of Southern California, 1999.

[16] Payton D., Kersey D., Kimble D., Krozel J., and Rosenblatt K. Do whatever works: A robust approach to fault-tolerant autonomous control. *Applied Intelligence*, 3:226–249, 1992.

[17] Saffiotti A., K. Konolige, and E. H. Ruspini. A multivalued-logic approach to integrating planning and control. *Artificial Intelligence*, 76(1-2):481–526, 1995.

[18] Hoffmann Frank. An overview on soft computing in behaviour based robotics. In *Proc. Joint 10th IFSA World Congress*, Istanbul, Juni 2003.

[19] Hoffmann Frank. Fuzzy behavior coordination for robot learning from demonstration. In *NAFIPS 2004*, Banff Canada, June 2004.

[20] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotic Research*, 5(1):90–98, 1986.

[21] Gregor Schöner, Michael Dose, and Christoph Engels. Dynamics of behavior: Theory and applications for autonomous robot architectures. *Robotics and Autonomous Systems*, 16, 1995.

[22] Sergio Monteiro and Estela Bicho. A dynamical systems approach to behavior-based formation control. In *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, Washington, DC, May 2002.

[23] Axel Steinhage and Gregor Schöner. Self-calibration based on invariant view recognition: Dynamic approach to navigation. *Robotics and Autonomous Systems*, 20:133–156, 1997.

[24] Axel Steinhage and Gregor Schöner. The dynamic approach to autonomous robot navigation. In *ISIE97, IEEE International Symposium On Industrial Electronics*, 1997.

[25] S. Goldenstein, D. M. Metaxis, , and E. W Large. Nonlinear dynamic systems for autonomous agent navigation. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence.* American Association for Artificial Intelligence., Menlo Park CA, 2000.

[26] Axel Steinhage. *Dynamical Systems for the Generation of Navigation Behavior*. PhD thesis, Ruhr-Universitaet Bochum, Germany, November 1997.

[27] Althaus Philipp. *Indoor Navigation for Mobile Robots: Control and Representations*. PhD thesis, Royal Institute of Technology, Stockholm, October 2003.

[28] kolmanovsky I. and McClamroch N. H. Development in Nonholonomic Control Problems. *IEEE Control Systems*, pages 20–36, December 1995.

[29] Fukao T., Nakagawa H., and Adachi N. Adaptive Tracking Control of a Nonholonomic Mobile Robot. *IEEE transactions on Robotics and Automation*, 16(5):609–615, October 2000.

[30] Kim Min-Soeng, Shin Jin-Ho, Hong Sun-Gi, and Lee Ju-Jang. Designing a Robust Adaptive Dynamic Controller for Nonholonomic Mobile

Robots under Modeling Uncertainty and Disturbances. *Mechatronics*, 13:507–519, 2003.

[31] Dongbing Gu and Huosheng Hu. Neural Predictive Control for a Car-like Mobile Robot. *International Journal of Robotics and Autonomous Systems*, 39(2–3), May 2002.

[32] Zhang Qiuju, J. Shippen, and J. Barrie. Robust backstepping and neural network control of a low quality nonholonomic mobile robot. *International Journal of Machine Tools and Manufacture*, 39:1117–1134, 1999.

[33] Fierro R. and Lewis F. L. Control of a Nonholonomic Mobile Robot Using Neural Networks. *IEEE Transactions on neural networks*, 9(4):589–600, July 1998.

[34] Fierro R. and Lewis F. L. Control of a nonholonomic mobile robot: Bachstepping kinematics into dynamics. *Journal of Robotic Systems*, 14(3):149–163, 1997.

[35] Watanabe K., Tang J., Nakamura M., Koga S., and Fukuda T. A fuzzy-gussian neural network and its application to mobile robot control. *IEEE Transaction on Control Systems Technology*, 4(2), March 1996.

[36] Rusu P., Petriu M., Whalen T. E, Cornell A., and Spoelder H. Behavior-Based Neuro-Fuzzy Controller for Mobile Robot Navigation. *IEEE Transaction on Instrum. and Measurment*, 52(4), August 2003.

[37] J.T. Connor, Martin R.D, and Atlas L.E. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5:240–254, March 1994.

[38] Kechriotis G.and Zervas E.and Manolakos E.S. Using recurrent neural networks for adaptive communication channel equalization. *IEEE Transactions on Neural Networks*, 5:267–278, March 1994.

[39] Chao-Chee Ku and Lee K.Y. Diagonal recurrent neural networks for dynamic systems control. *IEEE Transactions on Neural Networks*, 6:144–156, January 1995.

[40] Harter D.and Kozma R. Chaotic neurodynamics for autonomous agents. *IEEE Transactions on Neural Networks*, 16:565–579, May 2005.

[41] Zhu Quanmin and Lingzhong Guo. Stable adaptive neurocontrol for nonlinear discrete-time systems. *IEEE Transactions on Neural Networks*, 15:653–662, May 2004.

[42] Williams R. J. and Zipser D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.

[43] Perez-Ortiz J. A., Gers F. A., and D. Eck Schmidhuber J. Kalman filters improve lstm network performance in problems unsolvable by traditional recurrent nets. *Neural Networks*, 2:241–250, 2003.

[44] Schmidhuber Juergen. A fixed size storage o(n3) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248, 1992.

[45] Werbos Paul J. Backpropagation through time: What it does and how to do it. In *Proceedings of the IEEE*, pages 1550–1560, 1990.

[46] Hochreiter J. Untersuchung zu dynamischen neuronalen netzen. Master's thesis, Technische Universitt Mnchen, 1991.

[47] Hochreiter Sepp and Schmidhuber Jurgen. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[48] Graves A., Eck D., Beringer N., and Schmidhuber J. Biologically plausible speech recognition with lstm neural nets. In J. Ijspeert, editor, *Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pages 175–184, Lausanne, Switzerland, 2004.

[49] Douglas Eck and Juergen Schmidhuber. Learning the long-term structure of the blues. In J.R. Dorronsoro, editor, *Proceedings of International Conference on Artificial Neural Networks (ICANN 2002)*, pages 284–289, Madrid, 2002.

[50] Puskorius G.V. and Feldkamp L.A. Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks. *IEEE Trans. Neural Networks*, 5(2):279–297, 1994.

[51] Prokhorov D., Feldkamp L., and Tyukin I. Adaptive Behavior with Fixed Weights in Recurrent Neural Networks: An Overview. In *Proc. International Joint Conference on Neural Networks*, Honolulu,Hawaii, May 2002.

[52] Singhal S. and Wu L. Training multilayer perceptrons with the extended kalman algorithm. *Touretzky D. S. editor. Advances in Neural Information Processing Systems*, 1:133–140, 1989.

[53] Prokhorov D., Puskorius G., and Feldkamp L. *Dynamical Neural Networks for Control*. IEEE Press, 2001.

[54] Maass W.and Markram H. Temporal integration in recurrent microcircuits. In In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 1159–1163. MIT Press (Cambridge), 2 edition, 2003.

[55] Maass W., Natschlger T., and Markram H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560,, 2002.

[56] Buonomano D.V. and Merzenich M.M. Temporal information transformed into a spatial code by a neural network with realistic properties. *Science*, 267:1028–1030, Feb 1995.

[57] Jaeger H. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach. Technical Report 159, AIS Fraunhofer, St. Augustin, Germany, 2002.

[58] Maass W., Natschläger T., and Markram H. A model for real-time computation in generic neural microcircuits. In S.Becker, S.Thrun, and K.Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 229–236. MIT Press, 2003.

[59] Jaeger H. The 'echo state' approach to analysing and training recurrent neural networks. Technical Report 148, AIS Fraunhofer, St. Augustin, Germany, 2001.

[60] Jaeger H. and Haas H. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, April 2004.

[61] Cotter N.E.and Conwell P.R. Fixed-weight networks can learn. In *IJCNN International Joint Conference on Neural Networks*, volume 3, pages 553–559, June 1990.

[62] Vilalta R. and Drissi Y. A perspective view and survey of meta-learning. *Artificial intelligence Review*, 18:77–95, 2002.

[63] Schmidhuber J., Zhao J., and Schraudolph N. Reinforcement learning with self-modifying policies. In S. Thrun and L. Pratt, editors, *Learning to learn*, pages 293–309. Kluwer, 1997.

[64] Castiello C. and Castellano G.and Fanelli A.M. Designing a meta-learner by a neuro-fuzzy approach. In *IEEE Annual Meeting of the Fuzzy Information Processing*, volume 2, pages 893–898, June 2004.

[65] DesJardins M. and Gardon D. Evaluation and selection of biases in machine leaming. *Machine Leaning*, 20:5–22, 1995.

[66] Thrun S. Lifelong learning: A case study. Technical Report CMU-CS-95-208, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1995.

[67] Eric Horvitz Bennett Paul N., Susan T. Dumais. Inductive transfer for text classification using generalized reliability indicators. In *Proceedings of the ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*, Washington DC, U.S.A, August 2003.

[68] Schmidhuber J. On learning how to learn learning strategies. Technical report, Technical Report FKI-198-94, Fakultt fr Informatik, Technische Universitt Mnchen, 1994.

[69] Chan P.K and Stolfo S. Experiments on multistrategy learning by meta-learning. In *Proceedings of the International Conference on Information Knowledge Management*, pages 314–323, 1998.

[70] Wolpert D. H. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[71] Thrun S. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. Kluwer Academic Publishers, Boston, MA, 1996.

[72] Thrun S. A lifelong learning perspective for mobile robot control. In V. Graefe, editor, *Intelligent Robots and Systems*. Elsevier, 1995.

[73] Caruana Rich. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[74] Younger A. Steven, Conwell Peter R., and Cotter Neil E. Fixed weight on line learning. *IEEE Transaction on Neural networks*, 10(2):272–283, March 1999.

[75] Schmidhuber J. An 'introspective' network that can learn to run its own weight change algorithm. In *Third International Conference on Artificial Neural Networks*, pages 191–194, May 1993.

[76] Younger A.S., Hochreiter S., and Conwell P.R. Meta-learning with backpropagation. In *International Joint Conference on Neural Networks*, volume 3, pages 2001–2006, July 2001.

[77] Lo J. Adaptive vs. Accommodative Neural Networks for Adaptive System Identification. In *Proc. International Joint Conference on Neural Networks*, pages 1279–1284, 2001.

[78] Feldkamp L.A., Puskorius G.V., and Moore P.C. Adaptation from Fixed weight Dynamic Networks. In *Proc. IEEE International Conference on Neural Networks*, pages 155–160, Washington, 1996. IEEE.

[79] Feldkamp L.A., Puskorius G.V., and Moore P.C. Fixed weight Controller for Multiple Systems. In *Proc. IEEE International Conference on Neural Networks*, pages 773–778, Texas, June 1997. IEEE.

[80] Santiago Roberto. Context discerning multifunction networks: Reformulating fixed weight neural networks. In *International Joint Conference on Neural Networks*, July 2004.

[81] Feldkamp L., Prokhorov D., Eagen C., and Yuan F. Enhanced multistream kalman filter training for recurrent networks. In J. Suykens and J. Vandewalle, editors, *Nonlinear Modeling: Advanced Black-Box Techniques*, pages 29–53. Kluwer Academic Publishers, 1998.

[82] Mohamed Oubbati, Michael Schanz, and Paul Levi. Meta-learning for Adaptive Identification of Non-linear Dynamical Systems. In *Proc. Joint 20th IEEE International Symposium on Intelligent Control and*

13th Mediterranean Conference on Control and Automation, Limassol, Cyprus, June 2005. IEEE.

[83] Atiya A. F. and Parlos A. G. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE-NN*, 11(3):697, May 2000.

[84] Dong Wenjie an Wei Huo, S. K. Tso, and W. L. Xu. Tracking control of uncertain dynamic nonholonomic system and its application to wheeled mobile robots. *IEEE Transactions on robotics and automation*, 16(6), December 2000.

[85] Mohamed Oubbati, Michael Schanz, and Paul Levi. Recurrent Neural Network for Wheeled Mobile Robot Control. *WSEAS Transaction on Systems*, 3:2460–2467, August 2004.

[86] Mohamed Oubbati, Michael Schanz, and Paul Levi. A fixed-weight RNN Dynamic Controller for Multiple Mobile Robots. In *Proc. 24th IASTED International Conference on Modelling, Identification, and Control*, pages 277–282, Innsbruck, Austria, February 2005.

[87] Mohamed Oubbati, Michael Schanz, and Paul Levi. Kinematic and dynamic adaptive control of a nonholonomic mobile robot using a rnn. In *Proceedings of the 6th IEEE Symposium on Computational Intelligence in Robotics and Automation*, Helsinki, Finland, June 27-30 2005. IEEE.

[88] Williams R., B. Carter, P. Gallina, and G. Rosati. Dynamic model with slip for wheeled omnidirectional robots. *IEEE transactions on Robotics and Automation*, 18(3), June 2002.

[89] Chung J.H., B.Yi, W.K. Kim, and H. Lee. "the dynamic modeling and analysis for an omnidirectional mobile robot with three caster wheels". In *Proc. IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, September 2003. IEEE.

[90] Kalmár-Nagy Tamás, Raffaello D'Andrea, and Pritam Ganguly. "near-optimal dynamic trajectory generation and control of an omnidirectional vehicle". *Robotics and Autonomous Systems*, 46:47–64, 2004.

[91] Gerke M. and Hoyer H. Planning of optimal paths for autonomous agents moving in inhomogeneous environments. In *8th Int. Conf. Adv. Robot. (ICAR'97)*, 1997.

[92] Borgolte U. et al. Intelligent control of a semi-autonomous omnidirectional wheelchair. In *Proc. of the 3rd International Symposium on Intelligent Robotic Systems*, pages 113–120, Pisa, Italy, 1995.

[93] Mobile Robot Laboratory, Hans Moravec, Chuck Thorpe, Gregg Podnar, and Patrick Muir. Autonomous mobile robots, annual report. Technical Report CMU-RI-TR-86-04, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 1985.

[94] Fragois G. Pin and Stephen M. Killough. A new family of omnidirectional and holonomic wheeled platforms for mobile robots. *IEEE Transaction On Robotics and Automation*, 10(4), 1994.

[95] Diegel O. et al. Improved mecanum wheel design for omni-directional robots. In *Australien Conference on Robotics and Automation*, Auckland, November 2002.

[96] Wadi M. and H.Asada. Design and control of a variable footprint mechanism for holonomic omnidirectional vehicles and its application to wheelchairs. *IEEE Transaction On Robotics and Automation*, 15(6), 1999.

[97] West Mark and Haruhiko Asada. Design and control of ball wheel omnidirectional vehicles. In *ICRA*, pages 1931–1938, 1995.

[98] Mohamed Oubbati, Michael Schanz, Thorsten Buchheim, and Paul Levi. Velocity control of an omnidirectional robocup player with recurrent neural networks. In *Proceedings of the RoboCup International Symposium*, Osaka, Japan, July 2005.

[99] Mohamed Oubbati, Michael Schanz, and Paul Levi. Fixed-weight rnn adaptive controller for an omnidirectional robot. In *Proceedings of the 9th International Conference on Engineering Applications of Neural Networks (EANN05)*, Lille, France, August 24-26 2005.

[100] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of*

the IEEE Conference on Robotics and Automation, pages 1398–1404, Sacramento, California, April 1991.

[101] J. C Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[102] L.E Kavraki, P. Svestka, J.C Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[103] D. Jia and J. Vagners. Parallel evolutionary algorithms for uav path planning. In *Proceedings of the AIAA 1st Intelligent Systems Conference*, 2004.

[104] Amy J. Briggs, C. Detweiler, D. Scharstein, and A. Vandenberg-Rodes. Expected shortest paths for landmark-based robot navigation. In *Fifth International Workshop on Algorithmic Foundations of Robotics*, Nice, France, December 2002.

[105] Koenig S. and Likhachev M. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics and Automation*, 21(3):354–363., June 2005.

[106] Mohamed Oubbati, Michael Schanz, and Paul Levi. Neural fields for behavior-based control of mobile robots. In *Proc. 8th International IFAC Symposium on Robot Control (SYROCO 2006)*, Bologna, Italy, September 6-8 2006.

[107] Mohamed Oubbati, Michael Schanz, and Paul Levi. Neural fields for controlling formation of multiple robots. In *Proc. 3rd IEEE Conference On Intelligent Systems*, UK, September 4-6 2006.

[108] S. Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biol. Cybern.*, 27:77–87, 1977.

[109] M. A. Giese. Neural field model for the recognition of biological motion. In *Second International ICSC Symposium on Neural Computation*, Berlin, Germany, May 2000.

[110] H. Edelbrunner, U. Handmann, C. Igel, I. Leefken, and W. von Seelen. Application and optimization of neural field dynamics for driver

assistance. In *IEEE 4th International Conference on Intelligent Transportation Systems*, pages 309–314. IEEE Press, 2001.

[111] P. Dahm, C. Bruckhoff, , and F. Joublin. A neural field approach to robot motion control. In *Proceedings of the 1998 IEEE International Conference On Systems, Man, and Cybernetics*, pages 3460–3465, 1998.

[112] D. Eilam and I. Golani. Home base behavior of rats (rattus norvegicus) exploring a novel environment. *Behavioural Brain research*, 34:199–211, 1989.

[113] M.J. Mataric, M. Nilsson, and K.T. Simsarin. Cooperative multi-robot box-pushing. In *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems*, volume 3, Washington, DC, USA, 1995. IEEE Computer Society.

[114] T. Balch and R.C Arkin. Behavior-based formation control for multirobot teams. *IEEE Transactions on Robotics and Automation*, 14(6):926–939, December 1998.

[115] Jaydev P. Desai, James P. Ostrowski, and R. Vijay Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 17(6):905–908, December 2001.

[116] T. Buchheim, U.-P. Kaeppeler, R. Lafrenz, M. Oubbati, H. Rajaie, M. Schanz, F. Schreiber, O. Zweigle, and P. Levi. Team description paper 2005 cops stuttgart. In *RoboCup 2005*, Osaka, Japan., July 2005.

[117] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99–141, 2000.