Neurocontrol I: Self-organizing speed-field tracking

Csaba Szepesvári[†‡], and András Lőrincz[†]
{szepes,lorincz}@iserv.iki.kfki.hu

[†] Department of Photophysics
Institute of Isotopes of the
Hungarian Academy of Sciences
Budapest, P.O. Box 77, Hungary, H-1525

[‡] Bolyai Institute of Mathematics
University of Szeged
Szeged, Hungary, H-6720

August 15, 1996

Abstract

The problems of controlling a plant while avoiding obstacles and experiencing perturbations in the plants dynamics are considered. It is assumed that the plant's dynamics is not known in advance. To solve this problem a self-organizing artificial neural network (ANN) solution is advanced here. The ANN consists of various parts. The first part discretizes the state space of the plant and also learns the geometry of the state space. The learnt geometrical relations are represented by lateral connections. These connections are utilized for planning a speed field, allowing collision free motion. The speed field is defined over the neural represention of the state space and is transformed into control signals with the help of interneurons associated with the lateral connections: connections between interneurons and control neurons encode the inverse dynamics of the plant. These connections are learnt during a direct system inverse identification process by Hebbian learning. Theoretical results and computer experiments show the robustness of approach.

Contents

1 Introduction

Controlling a manipulator can be enormously complex from the point of view of an analytical approach since it requires the sequential computation of the location of the target, the path to be followed to reach the target, the inverse joint kinematics that satisfies the constraints of the path and the obstacles, the inverse joint dynamics and eventually the command series while meeting the demand of changes of the plant's dynamics and the environment.

Biological evidences strongly suggest that such a task can be solved with the help of learning. Effort along this route include various inverse system identification methods, such as the direct identification method Miller 1987; Kawato, Furukawa, and Suzuki 1987; Widrow, McCool, and Medoff 1978, the indirect method (that is based on the identification of the forward model) Jordan 1990; Werbos 1988; Widrow 1986 and the feedback-error learning method (when the errors generated by a previously fixed stabilizing feedback controller are used to train the inverse system identification model) Lewis, Abdallah, and Dawson 1993; Miyamoto, Kawato, Setoyama, and Suzuki 1988. For an overview see Dean and Wellman 1991; Miller, Sutton, and Werbos 1990; Narendra and Parthasarathy 1990 or Vemuri 1993. The focus of the present paper is a set of self-organizing artificial neural network (ANN) procedures that can be joined towards the solution of complex control issues. The self-organized principles are general, the computational examples serve illustrative purposes.

The organization of the article is as follows: In Section 2 the necessary background from control theory and the idea of path planning and speed field tracking are described. Section 3 concerns with feedforward control. The ANN that is capable of planning collision free speed fields, controlling the plant and learning the inverse dynamics is described as well as computational results are presented. This section is closed with a short discussion. Conclusions are drawn in Section 4.

The results presented here show that it is sufficient to learn to control the direction of motion while the speed of control can be treated as an independent variable when considering speed field tracking for collision free motion. This relaxes the learning problem. The proof for the learning of the inverse dynamics is also put forth here.

2 Preliminaries

2.1 Assumptions concerning the controlled plant

Let $\mathbf{R}^{m\times n}$ denote real $m \times n$ matrices. We say that a matrix $\mathbf{A}$ admits a generalized inverse[1] if there is a matrix $\mathbf{X}$ for which $\mathbf{AXA} = \mathbf{A}$ holds. It is well known that (i) $\mathbf{A}$ is nonsingular if and only if it has a unique generalized inverse and (ii) all the solutions of the linear equation $\mathbf{Ax} = \mathbf{b}$ have the form $\mathbf{x} = \mathbf{Xb} + (\mathbf{E} - \mathbf{XA})\mathbf{y}$ provided that the considered linear equation does in fact have a solution Ben-Israel and Greville 1974. Here $\mathbf{y}$ denoted an arbitrary vector of the appropriate dimensions. For convenience, the generalized inverse of a non-singular matrix $\mathbf{A}$ will be denoted by $\mathbf{A}^{-1}$.

Assume that the plant's equation is given in the following form Isidori 1989:

$$\dot{\mathbf{q}} = \mathbf{b}(\mathbf{q}) + \mathbf{A}(\mathbf{q})\,\mathbf{u} \tag{1}$$

where $\mathbf{q} \in \mathbf{R}^n$ is the state vector of the plant, $\dot{\mathbf{q}}$ is the time derivative of $\mathbf{q}$, $\mathbf{u} \in \mathbf{R}^m$ is the control signal, $\mathbf{b}(\mathbf{q}) \in \mathbf{R}^n$, and $\mathbf{A}(\mathbf{q}) \in \mathbf{R}^{n\times m}$. We assume that the domain (denoted by D) of the state variable $\mathbf{q}$ is compact and is simply connected; that $n \leq m$, and for each $\mathbf{q} \in D$ the rank of matrix $\mathbf{A}(\mathbf{q})$ is equal to n; that is, the matrix is nonsingular. As a consequence the plant is strongly controllable. In this case the inequality $n < m$ means that there are more independent actuators than state vector components, i.e., the control problem is redundant. Another kind of redundancy, or ill-posedness occurs when $n > m$ in which case even $\mathbf{A}^{-1}$ is non-unique.

Further, we assume that both of the matrix fields, $\mathbf{A}(\mathbf{q})$ and $\mathbf{A}^{-1}(\mathbf{q})$ are differentiable w.r.t. $\mathbf{q}$ (differentiation is assumed to be extended to matrix fields in the usual way Lovelock and Rund 1975).

2.2 Speed field tracking

One way to obtain a closed-loop control task is to consider the speed field tracking problem. This is defined as follows: Let $\mathbf{v} = \mathbf{v}(\mathbf{q})$ be a fixed n dimensional vector field over D. The *speed field tracking task* is to find the static state feedback control $\mathbf{u} = \mathbf{u}(\mathbf{q})$ that solves the equation

$$\mathbf{v}(\mathbf{q}) = \mathbf{b}(\mathbf{q}) + \mathbf{A}(\mathbf{q})\mathbf{u}(\mathbf{q}). \tag{2}$$

[1]Sometimes it is called the pseudo-inverse, or simply the inverse of matrix $\mathbf{A}$.

More conventional tasks, such as the *point to point control* and the *trajectory tracking* tasks cannot be exactly rewritten in the form of speed field tracking. Speed field tracking is non-typical in the control literature, but arises naturally if we consider path planning tasks Connolly and Grupen 1993; Fomin, Szepesvári, and Lőrincz 1994; Lei 1990. The importance of speed field tracking for path planning can be summarized as follows: In the case of point to point control the task is to find a control that moves the plant from a given initial state $(\mathbf{q}_i, \dot{\mathbf{q}}_i)$ into a prespecified final state given by $\mathbf{q}_f$ and $\dot{\mathbf{q}}_f = 0$, the control signal being a function of time. Point to point control is ill-posed as there are an infinite number of paths to $(\mathbf{q}_f, \dot{\mathbf{q}}_f = 0)$. However, if one requires "collision free" motion, i.e., when the plant should not enter a so called (stationary) obstacle region, then a huge number of solutions become unacceptable. One way to ensure collision free motion is to design a collision free path as a function of time, $\mathbf{q}_d(t)$, and track this path as closely as possible. In this way one arrives at the trajectory tracking task when a trajectory is given and the aim of the control is to find a feedback control law which is able to impose on the error $\mathbf{q}(t) - \mathbf{q}_d(t)$ a behavior which asymptotically decays to zero as time tends to infinity. For convenience, it is usually assumed that the desired reference trajectory is not just a fixed function of time but, rather, coincides with the output of some autonomous dynamical system. Speed field tracking may be considered as a special case of trajectory tracking provided that the distinguished autonomous system is the plant itself controlled by an optimally designed state feedback control law. The major difference between speed field tracking and trajectory tracking is caused by the fact that a speed field is given as a function of state while a trajectory is given as a function of time. Consequently speed field tracking is more robust against state perturbations. This can be important if it is critical to ensure collision free motion. Note that trajectory tracking may result in collision if the actual and the desired states of the plant differ sufficiently. Often it is hard to exclude the possibility of such differences because of unforeseen disturbances. With speed field tracking there is no such problem since the speed field determines the motion as a function of state rather than as a function of time.

Speed field tracking is robust in the following sense: Assume that the speed field $\mathbf{v} = \mathbf{v}(\mathbf{q})$ results in collision free motion. Then for any scalar field $\lambda = \lambda(\mathbf{q})$ for which $0 < \inf_{\mathbf{q}} \lambda(\mathbf{q})$ and $\sup_{\mathbf{q}} \lambda(\mathbf{q}) < \infty$ the speed field $\mathbf{v}' = \lambda(\mathbf{q})\mathbf{v}(\mathbf{q})$ results in collision free motion, too. In order to see it note

that the integral curves of the equation $\dot{\mathbf{q}} = \mathbf{v}(\mathbf{q})$ and $\dot{\mathbf{q}} = \mathbf{v}'(\mathbf{q})$ are the same since the integral curves of these equations are completely determined by their unity length tangentials. These normalized tangentials, however, are just the same for the two speed fields. This property is important since this means that given a speed field that guarantees collision free motion one can freely redefine the speed of following this field.

Speed field design for collision free control is the subject of current research Hwang and Ahuja 1992. An efficient but also quite peculiar way of constructing the speed field is to compute the stationary flow of a well designed diffusion over the state space Connolly and Grupen 1993; Glasius, Komoda, and Gielen 1994; Keymeulen and Decuyper 1992; Lei 1990; Morasso, Sanguineti, and Tsuji 1993; Tarassenko and Blake 1991. The neural architecture that is capable of designing a discretized speed field is described in Section 3. Another method for constructing the speed field is the potential field method Locano-Pèrez and Wesley 1979. Artificial potential fields, however, may have deceptive local minima.

2.3 Inverse dynamics

Given the plant's dynamics by Equation (1) the inverse dynamics of the plant is given as follows:

$$\mathbf{p}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{A}^{-1}(\mathbf{q})\Big(\dot{\mathbf{q}} - \mathbf{b}(\mathbf{q})\Big) + \Big(\mathbf{E} - \mathbf{A}^{-1}(\mathbf{q})\mathbf{A}(\mathbf{q})\Big)\mathbf{y}(\mathbf{q}, t), \tag{3}$$

where $\mathbf{y} = \mathbf{y}(\mathbf{q}, t)$ is an arbitrary function. Of course, the control signal

$$\mathbf{u}(\mathbf{q}) = \mathbf{p}(\mathbf{q}, \mathbf{v}(\mathbf{q}))$$

solves the speed field tracking control task given by Equation (2). *The main value of the inverse dynamics is given by* $\mathbf{y}(\mathbf{q}, t) = 0$, *i.e., by*

$$\mathbf{p}(\mathbf{q}, \mathbf{v}) = \mathbf{A}^{-1}(\mathbf{q})\Big(\dot{\mathbf{q}} - \mathbf{b}(\mathbf{q})\Big). \tag{4}$$

This assumption simplifies the calculations and is justified later.

2.4 Speed field planning

In this section we review the works Lei 1990; Keymeulen and Decuyper 1992; Connolly and Grupen 1993; Glasius, Komoda, and Gielen 1994; Marshall

and Tarassenko 1994 that served as the origin of our speed field planning neural network. The neural *spreading activation* (SA) method we consider can be viewed as the discretization of the following diffusion like differential equation:

$$\dot{\varphi} = \Delta\varphi + I, \tag{5}$$

where $\varphi = \varphi(\mathbf{q}, t)$ is the activity, $I = I(\mathbf{q})$ is the external flow and

$$\Delta = (\partial^2/\partial x_1^2 + \partial^2/\partial x_2^2 + \ldots + \partial^2/\partial x_n^2)$$

is the Laplacean operator. Let us denote the stationary solution of (5) by $\varphi^* = \varphi^*(\mathbf{q})$. Then the equation of motion of the plant is given by

$$\dot{\mathbf{q}} = \kappa\nabla\varphi^*(\mathbf{q}), \tag{6}$$

where κ is a positive constant, i.e., the speed field to follow is given by $\mathbf{v}(\mathbf{q}) = \kappa\nabla\varphi^*(\mathbf{q})$.

The external flow is such that the plant moves from the start to the target state: it is determined by the actual state of the plant taking the value of 1 at the actual state and the value of -1 at the target state, otherwise being zero. This way activity flows from the actual state towards the target state. The boundary conditions of Equation (5) may be chosen to ensure collision free paths. Two methods typically used that guarentee the flow to avoid the forbidden zone. The Neumann type of boundary condition requires the normal component of the flow with respect to the boundary of F to be zero:

$$\frac{\partial\varphi}{\partial\mathbf{n}}\Big|_{\partial F} = 0. \tag{7}$$

As a result the flow avoids the obstacle space. The Dirichlet type of boundary condition sets the flow level along the boundary of free space constant:

$$\varphi\Big|_{\partial F} = c. \tag{8}$$

The Dirichlet boundary condition can result in a tendency for the plant to depart from the forbidden zone. The Neumann condition, on the other hand, specifies that the flow is tangential to the boundary of the forbidden zone and the resulting motion stays close to the forbidden zone. Here we note that by using the linear combination of the Neumann and Dirichlet boundary conditions the tendency to depart from the forbidden zone can continuously be balanced.

3 Feedforward Control

The neurocontroller, which is described below, was suggested in Fomin, Szepesvári, and Lőrincz 1994; Szepesvári and Lőrincz 1995 for closed-loop feedforward sensorimotor control. The basis of the neurocontroller is the path planning dynamics that is descibed in the previous section. We have extended this dynamics to include the learning of an approximate inverse dynamics model of the plant. Now, a brief overview of the controller is given.

Let us first consider the path planning part of the neurocontroller (see Fig. 1). The path planning problem is given in terms of discretization point occupancies. Any discretization point, called *spatially tuned neuron*, can be occupied by an obstacle, the plant, or the target. It is also possible that more than one discretization point is occupied by an object. This results in a coarse coded, distributed representation of the object that in turn results in smoother control signals. Between *neighbouring* discretization points laterally oriented *geometrical* connections may be developed. It can be shown that such a system can approximate the geometry of the external world under suitable conditions Martinetz 1993; Szepesvári and Lőrincz 1996. The geometrical connections can be used to spread the activation within the neural layer. When the activity spreading on the discretization system settles we say that an activity field is formed. We call this the equilibrium activity map. The plant should move along the "gradient" of this activity map. (Here, and below we consider the neural network as a numerical approximation of a continuous system. If the concepts are used with care then one can talk about the gradient field in the discretized system, i.e., the approximation of the corresponding quantity in the continuous system. The expression, directional derivative will also be used in this way.) The activation spreading equation that forms this activity map is the discretization of Equation (5). The obstacle avoidance is achieved by setting up the appropriate boundary conditions, our case by forbidding the activity to spread along the lateral connections of the corresponding neurons thus approximating the Neumann boundary condition around the obstacles. If the gradient of the equilibrium map is followed it results in a path from the plant's actual position to the goal position. For on-line motion control the activity map should be continuously upgraded. This is important if either the obstacles or the goal is moving, or the controller or the sensors are imperfect. For continuous motion the changes of the equilibrium activity map are differential and thus the

relaxation time of the spreading activation model is a differential quantity. This enables fast, on-line path planning.

It is also a nontrivial task to follow the gradient of the equilibrium activity map. This task, namely *tracking a prescribed speed field* is described in Section 2.2. Solving this task requires a knowledge of the inverse dynamics of the plant. However, as it was observed above, in order to ensure collision free motion it is enough to follow a proportional speed field, where the proportionality can be a function of the state provided that the proportionality has a positive lower bound and a finite upper bound. This eases the learning problem since it suffices to know a mapping that is proportional to the inverse dynamics mapping. Such a proportional mapping is called the Position-Direction to Action (PDA) mapping. It has been shown in Fomin, Szepesvári, and Lőrincz 1994; Szepesvári and Lőrincz 1995 that the realization and learning of PDA mapping can be solved by simple Hebbian learning and by extending the path planning architecture by adding two new neuronal layers: the path planner neural net is equipped with interneurons and control (command) neurons (see Fig. 1).

Control neurons should emit the control signal that moves the plant along the gradient. Interneurons are situated at lateral connections (to each connection there correspond two directives and thus two interneurons) and are connected to the control command neurons by adaptive connections. Equivalently, interneurons can be considered to store control commands. The working mechanism of the neurocontroller is as follows: An interneuron is enabled to "fire" only if it is in the neighbourhood of the plant's state represented on the discretization layer. This localization (similar to CMAC and Radial Basis Function methods) enables state dependent non-linear inverse dynamics to be realized. The firing of an interneuron is proportional to the extent of flow along the corresponding connection (i.e., the firing of an interneuron is an approximation of the directional derivative of the steady state activity map). Every firing interneuron sends its control command multiplied by its firing to the control neurons. These neurons sum up their incoming activities and emit the computed value. This procedure approximates the PDA mapping provided that the control commands of any interneuron move the plant along the direction of the corresponding connection. It has been shown that this approximation is exact in the limit when the fineness of the discretization approaches zero and if the local neighbourhood of any given discretization point is spanned by the direction vectors corresponding to the neighbouring

interneurons of the discretization point under consideration.

The adaptation of the weights between interneurons and control command neurons is based on a general inverse identification scheme that utilizes associative Hebbian learning: a randomly chosen control command and the interneuron signals are associated with Hebbian learning, where the interneuron signals are computed from the path planning problem where the "plant-position" and "goal-position" are the plant's initial position and the position after the execution of the randomly chosen control command, respectively. It has been shown that in this way the neurocontroller is capable of learning the correct control commands and further it learns a proportional mapping to the main value of the inverse dynamics (see Equation 4). The advantage of self-organized associative learning is that it can not be trapped in local minima. However, its disadvantage is that exhaustive sampling of a high dimensional control space can be very time consuming.

In the next few sections we describe the working and learning of this neurocontroller in details.

3.1 Speed Field Planning by a Neural Network

In this section we review previious works that solve Equation (5) by discretizing the state space of the plant Connolly and Grupen 1993; Glasius, Komoda, and Gielen 1994; Keymeulen and Decuyper 1992; Lei 1990; Morasso, Sanguineti, and Tsuji 1993; Tarassenko and Blake 1991. Concepts are expressed in artificial neural network terms in order to emphasize the highly parallel and localized nature of the involved computations.

Discretization points, or grid points, or *discretizing neurons* are evenly distributed in the state space. It is assumed that every discretizing neuron i has a position $\mathbf{c}_i$ in the state space. The response of a neuron to a state space vector is assumed to depend on its position. With the help of the discretizing neurons the path planning problem may be approximated by distinguishing four types of neurons: *target neurons* correspond to the target state, *start neurons* correspond to the start state, *active neurons* correspond to the free-space, and *inactive neurons* correspond to the forbidden region of the state space. It is assumed that there is only one target and one start neuron and that the set of active and inactive neurons is disjunct. The degree to which every neuron participates in any of the above classes is either 0 or 1. Thus we may speak about target and start activities as well as obstacle activities.

The diffusion process (Equation (5)) with the Neumann type of boundary condition is simulated on the discretizing layer by the following equation:

$$\dot{\sigma}_i = I_i + \sum_{k \in N_i \cap F} (\sigma_k - \sigma_i), \quad i \in F \tag{9}$$

where F is the set of active neurons (corresponding to the free space) and N_i denotes the set of neighbouring grid points of grid point i and σ_i is the discretized version of the activity flow φ at position $\mathbf{c}_i$. The external signal I_i takes the values of 0, 1 or -1. $I_i = 1$ if neuron i is the start neuron, $I_i = -1$ if neuron i is the target neuron and $I_i = 0$, otherwise. Equation (9) can be interpreted as *spreading of activation* (SA): activation σ_i can spread between neighbouring neurons provided that they both belong to the free space region. This way activation avoids obstacle regions. The external signal I_i corresponds to the source and sink flows. If the source and sink flows are equal ($\sum_i I_i = 0$) then the total activity of the network remains constant during the process ($\sum_i \dot{\sigma}_i = 0$).

The next step of the path generation procedure is the determination of the equilibrium activities of (9). Based on the equilibrium activity map one may adopt the following naive path planning procedure: At any time step the subsequent state of the plant is defined as the neighbouring position of the start neuron with the steepest activity drop. Assume, that the index of the start neuron is i. Let l be the index of its neighbouring neuron satisfying

$$\sigma_i - \sigma_l = \max_{k \in N_i \cap F} (\sigma_i - \sigma_k). \tag{10}$$

The next state of the plant is given by $\mathbf{c}_l$.

After the plant is moved to state $\mathbf{c}_l$ the procedure is repeated: the path planning task is transformed onto the neuronal layer, SA starts and settles and the next state is determined according to the "gradient" of the stationary activation. Note, that the initial activities of an SA process may be taken as the relaxed activities of the previous step. In this case the finer the discretization the shorter the expected relaxation time that reaches zero in the continuous limit. This procedure was termed naive because of two reasons: (i) with little sophistication the move to the neighbouring neuron may be smoothened and (ii) the move to the neighbouring neuron is not know yet and should be the subject of learning.

3.1.1 The feedforward controller

In this section we introduce the architecture and the functioning of the proposed control network. First the SA model of the previous section is extended to plan smooth trajectories in a natural fashion. Then the mathematical background of the combination of the control equation (Equation (1)) and the path generation equation (Equation (6)) are given. These results motivate the extension of the architecture by interneurons, control neurons and control command connections. The functioning of the network is designed in a way that the calculations remain local and parallel and the output of control neurons provide directly the control signal of the plant. Finally we show how direct associative learning can be used to learn the optimal control command vectors.

3.1.2 Coarse coding and gradient estimation

Smooth trajectories are desirable in many applications. In the ANN model discussed above there are two sources of non-smooth trajectories:

1. Discretizing neurons with binary outputs.

2. One state of the plant corresponds to one grid point.

In fact, the second assumption implies the first and if the second assumption is relaxed then the first assumption means an ambiguous (or inaccurate) problem representation. This problem will be circumvented as follows:

The first assumption is relaxed by enabling the neurons to develope continuous response signals, i.e. by enabling coarse coding. This way we get a fuzzy representation of the path planning problem being advantegous in many respects Rumelhart, Hinton, and Williams 1986. As a consequence, the "fuzzy set" of start, target, active and inactive neurons may overlap. For safety reasons we classify neurons having above threshold obstacle activities as inactive neurons. Other neurons are considered active.

Since start and target neurons are no longer unique the external flow I_i will be determined as follows:

$$I_i = s_i/s - t_i/t$$

where s_i and t_i are the continuous start and target activities of neuron i, respectively and $t = \sum_{i \in F} t_i$ and $s = \sum_{i \in F} s_i$ are flow normalizing factors.

To relax the second assumption we reconsider the continuous equation of motion of (6). According to this equation we have to determine the speed vector of motion given as the gradient of the stationary flow at the start position.

The gradient can be approximated by directional derivatives of the stationary flow. To this end let us introduce the concept of geometry vectors. The geometry vector between neuron i and j is the vector that points from the position $\mathbf{c}_i$ of neuron i towards the position $\mathbf{c}_j$ of neuron j: $\mathbf{g}_{ij} = \mathbf{c}_j - \mathbf{c}_i$. Let σ_i denote the stationary activity at node i. The gradient of the stationary activity flow at node i is approximated by

$$\mathbf{d}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{g}_{ij}, \tag{11}$$

where

$$I_{ij} = (\sigma_j - \sigma_i) w_{ij} \tag{12}$$

is the approximation of the directional derivative of σ with respect to $\mathbf{g}_{ij}$ at the point $\mathbf{c}_i$, where

$$w_{ij} = \frac{1}{||\mathbf{c}_j - \mathbf{c}_i||^2}, \quad j \in N_i. \tag{13}$$

The w_{ij} values are defined only for neighbouring neurons and are called the strength of *neighbouring connection* between neuron i and j.[2] The gradient of the flow at the state of the plant is approximated by the gradients of the flow at discretizing points (neurons) weighted by the coarse coded activities of the state of the plant:

$$\mathbf{d} = \sum_{i \in F} s_i \mathbf{d}_i, \tag{14}$$

Note, that in equation (11) it is necessary to restrict the summation for active nodes (elements of F) since activities, and thus I_{ij} values are only defined for active nodes. The I_{ij} values may be interpreted as the activity flow along the neighbouring connection between neuron i and j.

3.1.3 Following the gradient

In order to realize Equation (2) in practice we have to represent the inverse dynamics of the plant in a suitable way. First let us fix an arbitrary point

[2] Equation 9 should be modified accordingly by replacing the terms of the summation with the I_{ij} values.

$\mathbf{q}$ in the space. This point may be chosen as a discretization point. Let $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbf{R}^n$ denote k "direction" vectors ($k \geq n$). One might think of the geometrical vectors belonging to a discretization point. Assume, that the contol vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k \in \mathbf{R}^m$ satisfy the equalities

$$\mathbf{v}_i = \mathbf{b}(\mathbf{q}) + \mathbf{A}(\mathbf{q})\mathbf{a}_i, \quad i = 1, \ldots, k. \tag{15}$$

Assume, that the k direction vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$ span the n dimensional space. We propose that the k control vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k$ are sufficient for controlling the plant at the state $\mathbf{q}$. To show this, assume that the plant is to be moved into the direction $\mathbf{d}$ from the point $\mathbf{q}$ and $\mathbf{d}$ is expressed as

$$\mathbf{d} = \sum_{i=1}^{k} \alpha_i \mathbf{v}_i, \quad \text{with} \sum_{i=1}^{k} \alpha_i = 1. \tag{16}$$

Note, that if there are at least $n+1$ vectors among the vectors $\mathbf{v}_i$ that are affine independent (i.e., any n vector span $\mathbf{R}^n$), then coefficients that satisfy $\sum_i \alpha_i = 1$ may be found. Let us consider the control vector

$$\mathbf{a} = \sum_{i=1}^{k} \alpha_i \mathbf{a}_i. \tag{17}$$

Substituting (17) into equation (1) we have that the control vector $\mathbf{a}$ yields the speed vector $\mathbf{d}$.

3.1.4 Computing the gradient and motion control

Assume, that we are given a path planning problem and the recurrent network has already relaxed in a stationary state. One may compute then the control signal in the following fashion: The speed vector of the plant must correspond to the gradient of the stationary flow at the start state. In section 3.1.2 it was shown that the gradient of the flow at the start state can be estimated by $\mathbf{d} = \sum_{i \in F} s_i \mathbf{d}_i$, where $\mathbf{d}_i$ is the approximated gradient at neuron i: $\mathbf{d}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{g}_{ij}$, where $I_{ij} = w_{ij}(\sigma_j - \sigma_i)$. According to the previous section if one is given the control vectors $\mathbf{a}_{ij}$, $j \in N_i$, satisfying

$$\mathbf{g}_{ij} = \mathbf{b}(\mathbf{c}_i) + \mathbf{A}(\mathbf{c}_i)\mathbf{a}_{ij} \tag{18}$$

then

$$\mathbf{a}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{a}_{ij} \tag{19}$$

moves the plant into the direction $\mathbf{d}_i$ provided that the plant is in state $\mathbf{c}_i$.[3] Taking into account the coarse coding of the state of the plant, i.e., that the state of the plant is given as the blob $\{ s_i \}$ we get that the control vector is approximated by

$$\mathbf{a} = \sum_i s_i \mathbf{a}_i. \tag{20}$$

Considerations (19) and (20) fit well to the recurrent architecture that compute the stationary flow. Let us extend the architecture by control neurons and interneurons. Control neurons provide control signals and are connected to interneurons via command connections. Interneurons correspond to neighbouring connections and monitor the activities that flow along the connections and provide proportional outputs with those, i.e. the output of interneuron (i, j) is given by $s_i I_{ij}$. Let the command connection that starts from interneuron (i, j) and ends on control neuron k be the kth component of $\mathbf{a}_{ij}$. Then the motion planning and execution procedure is the following:

Step 1. Develope the coarse coding of the path planning task on the recurrent network.

Step 2. Compute the stationary flow by activation spreading.

Step 3. Interneurons compute the directional derivatives of the flow weighted by the coarse coding activities of the actual state of the plant.

Step 4. Interneurons send their outputs through the command connections to the control neurons. The sum of received activities is the control signal.

The architecture is shown in Figure 1.

3.1.5 Direct, associative identification of the inverse dynamics

The aim of this section is to show that action vectors satisfying Equation (19) may be learnt. The first observation is that the stationary activation of the SA flow can provide an immediate reinforcement signal: the level of the SA at the start state increases (decreases) as the plant moves closer (farther)

[3] In Equation (19) the I_{ij} coefficients should have been normalized but according to our numerical experiments Equation (19) can work equally well.

relative to the previously defined and fixed target. The stationary SA flow differences may be used to evaluate the current motion combination. SA thus facilitates reinforcement learning. It may be worth noting that target oriented motion is possible even without learning provided that motions are invertible: motions that bring the plant farther should be undone while motions that bring the plant closer can be accepted.

The learning scheme we propose below is similar to direct inverse modeling Widrow, McCool, and Medoff 1978; Psaltis, Sideris, and Yamamura 1988; Grossberg and Kuperstein 1986: The control signal and the movement produced by the plant in response to the control signal provide the data for learning. However, our method is not a variation of error back-propagation (or in control theoretical terms we are not using the method of variations). The main point of the algorithm is that the movement is represented by the stationary flow corresponding to the initial state of the plant as the source and the state of the plant after the movement as the target state. This way the algorithm is fully self-organized and self-contained. The learning of the additive term $\mathbf{b}(\mathbf{q})$ of the inverse dynamics is not detailed here: it can be easily learnt by using the plant's zero dynamics; i.e., by setting $\mathbf{u} = 0$ and learning the resulting movements. In below only the learning of the multiplicative term, $\mathbf{A}(\mathbf{q})$ will be considered. The algorithm is as follows:

Step 1. Develop the coarse coding that corresponds to the state of the plant. Store it as start activities.

Step 2. Choose a random control signal and feed it into the plant.

Step 3. Compute the coarse coding of the resulting state of the plant and use it as the target activities.

Step 4. Compute the stationary flow according to these start and target activities.

Step 5. Associate the control signal to interneurons weighted by the output of the interneurons.

In Step 5. the signal Hebbian learning rule may be used, i.e.

$$\Delta \mathbf{a}_{ij} = \alpha_{ij}(s_i I_{ij} \mathbf{a} - \mathbf{a}_{ij}), \tag{21}$$

where $0 < \alpha_{ij} < 1$ is the learning rate of interneuron (i, j). The learning rate can be time dependent or stationary. It is reasonable to choose a Robbins-Monro type time dependence Robbins and Monro 1951; Wasan 1969 in order to ensure the convergence of $\mathbf{a}_{ij}$ to the average of learning samples with respect to the input distribution and also to maintain adaptivity forever. However, in this case adaptivity may become extremely slow by time. If the learning rate is kept constant then adaptivity may be kept above a predefined level. In this case $\mathbf{a}_{ij}$ may be considered as a stochastic variable with mean given by the sample average and deviation magnitude proportional to $\sqrt{\alpha_{ij}}$ Amari 1967 after enough training epochs have passed.

Another possible learning method would be to store the "best action vector so far". However, this method may be sensitive to state errors, while the Hebbian learning does average and can handle noisy inputs too.

Yet another possible method is to restrict learning to the immediate neighbourhood of the neuron with the highest s_i activity, and further to the interneuron that has the highest flow I_{ij}, that is

$$\Delta \mathbf{a}_{ij} = \begin{cases} \alpha_{ij}(\mathbf{a} - \mathbf{a}_{ij}); & \text{if } s_i = \max_k s_j \text{ and } I_{ij} = \max_l I_{il}, \\ 0; & \text{otherwise.} \end{cases} \tag{22}$$

Learning rule (21) can be considered as the "soft" version of (22). For this latter rule we can prove that the limit of the $\mathbf{a}_{ij}$ vectors satisfy (18) and thus the algorithm can learn the inverse dynamics of any plant up to the precision of discretization.

Again, if the time dependence of the α_{ij} values satisfy $\sum_t \alpha_{ij}(t) = \infty$ and $\sum_t \alpha_{ij}^2(t) < \infty$, where t is incremented only if interneuron (i, j) learns, then the convergence of $\mathbf{a}_{ij}$ is guaranteed. The limit of action vector corresponding to interneuron (i, j) can be written in the form

$$\overline{\mathbf{a}} = \int_W dP(\mathbf{x}) \int_{Y(x)} u(\mathbf{x}, \mathbf{y}) dP(\mathbf{y}|\mathbf{x}),$$

where $u(\mathbf{x}, \mathbf{y}) = \mathbf{A}^{-1}(\mathbf{x} - \mathbf{b}) + (\mathbf{E} - \mathbf{A}^{-1}\mathbf{A})\mathbf{y}$ with $\mathbf{A} = \mathbf{A}(\mathbf{c}_i)$, $\mathbf{b} = \mathbf{b}(\mathbf{c}_i)$ and W is the set of $\mathbf{x}$ direction vectors for which the geometry connection (i, j) is the "winner", and $Y(\mathbf{x})$ represents an arbitrary measurable set. Substituting the expression for $u(\mathbf{x}, \mathbf{y})$ in the above equation yields

$$\overline{\mathbf{a}} = \mathbf{A}^{-1}(\overline{\mathbf{x}} - \mathbf{b}) + \left(\mathbf{E} - \mathbf{A}^{-1}\mathbf{A}\right) \int_W dP(\mathbf{x}) \int_{Y(\mathbf{x})} \mathbf{y} dP(\mathbf{y}|\mathbf{x}),$$

where $\overline{\mathbf{x}} = \int_W \mathbf{x} dP(\mathbf{x})$. From this follows that if W is symmetrical w.r.t. $\mathbf{g}_{ij}$ (e.g. the disretization is regular), then $\overline{\mathbf{x}} = \mathbf{g}_{ij}$. Thus in this case $\overline{\mathbf{a}}$ satisfies Equation (18). Moreover, if the sampling of $\mathbf{y}$s is also symmetrical for each given $\mathbf{x}$, that is if $Y(\mathbf{x})$ is centrally symmetric, then the second term in the above equation disappears, too.

3.2 Computational results

In the examples presented hereafter the state space is the two dimensional rectangle in $[0,1] \times [0,1]$, while the control system of the plant has four components. Thus we treat a redundant control problem: the degrees of freedom in the motor command space is higher than the degrees of freedom in the task space. As pointed out by Jordan such redundancy can not be solved by direct error back-propagation: there are infinite possible combinations of command errors that would lead to the same error expressed in task coordinates Jordan 1990. However, as we will see our system is capable of solving this redundant tasks.

The presented control problems are simple *sensory-motor control* problems: the state space is represented through a sensor system. A simulated camera, i.e., a pixel discretization provides the input for the system. The discretizing neurons work on the "image space" of the simulated camera instead of the state space, However, the product of the two discretizations, i.e. the product of the discretization developed by the camera (as a function from the state to the image space) and the discretization developed by the neurons (as a function from the image space to neuronal activities), is itself a discretization of the state space. Since neighbouring connections reflect neighbourships in the state space – i.e., the discretization is topographic Szepesvári and Lőrincz 1996 – the extra discretization step does not affect the working of the model (apart from resolutional effects).

We present two control problems for comparison: one with linear and another one with nonlinear $\mathbf{q}$ dependence. In the case of the linear control problem the control components alone move the plant respectively towards north, east, south and west. The control equation of the plant is given by

$$\dot{\mathbf{q}} = F\mathbf{u},$$

where $\mathbf{u} \in \mathbf{R}^4$, $\mathbf{q} \in [-1,1]^2$ and

$$F = F_0 = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

In the second example F is position dependent in a non-linear fashion:

$$\dot{\mathbf{q}} = F(\mathbf{q})\mathbf{u}, \tag{23}$$

$$F(\mathbf{q}) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} F_0,$$

where $\alpha = \alpha(\mathbf{q}) = 2\alpha_{\max}(0.5 - d)$ provided that $d = \sqrt{(q_1 - 0.5)^2 + (q_2 - 0.5)^2} < 0.5$ and $\alpha = 0$ otherwise. In the experiments we used the setting $\alpha_{\max} = \frac{\pi}{2}$. That is the speed vector of motion is the rotated speed vector of the linear case. The rotation angle is state dependent: the rotation is $\pi/2$ at the centre point $(0.5, 0.5)$ and decreases with the distance from the center. The rotation is zero outside the circe with radius 0.5.

The purpose of the experiments is twofold. To demonstrate

completeness: the model is capable of path planning and *execution* in the case of non-trivial, non-linear control problems

learning capabilities: the learning algorithm is capable of representing non-linear inverse dynamics

Complex path planning tasks are not presented here since previous works Lei 1990; Keymeulen and Decuyper 1992; Connolly and Grupen 1993; Glasius, Komoda, and Gielen 1994 have already demonstrated the potential of the method as a path planning procedure.

For visualization we present a result of the diffusion system. Initial activities are shown in Figure 2. Start activities are negative and target activities are positive. The target and the plant are placed in the opposite corners. The relaxed activity field is depicted in Figure 3.

The start positions in the the first two experiments were designated on the upper and on the right sides of the rectangle. The target position was in the lower left corner. The motion learning was performed in free space. The left and middle subfigures of Figure 4 show motion trajectories in the case of linear and non-linear dynamics, respectively. The paths are curved

around the edges. This "edge effect" is the result of the motion control; motion vectors and activation spreading are not balanced at the borders. Note that the model gives higher precision than the quality of discretization: the discretization in these cases was very rough: if one consideres the state space as a square of side length 1 meter then a neuron represents roughly a 20 cm by 20 cm area. The discretization and the geometry representation were developed in a self-organizing fashion Szepesvári, Balázs, and Lőrincz 1994. The right hand side figure shows the results with an obstacle in the workspace (the dynamics was linear). As it has already been discussed the path planning method allows the motion planning in such spaces without any further training. Note, that some of the motion trajectories hit the obstacle. This is clearly the effect of rough discretization.

We would like to emphasize that as a result of coarse coding and linear gradient estimation the speed vector field is continuous, i.e., the trajectory of the plant is smooth. This is demonstrated in Figure 5 showing the the control signals vs. time during the same course of action. The figure corresponds to learnt control vectors and the non-linear plant.

3.3 Discussion of feedforward control

3.3.1 Workspace vs. configuration space

In the previous section we have already touched the question of sensory-motor control. In sensory-motor control a sensor space serves as the input to the algorithm instead of the state space. In the treated example the sensor space was a discretization of the state space. In the general case, however, the connection between the sensor space and the state space is non-trivial. Consider, for example, a robot manipulator with more than 3 degrees of freedom (say k) in a 3 dimensional space. Assume that two cameras monitor the end-point of the manipulator. Then the state space is k-dimensional while the workspace and the image manifold are both 3 dimensional. Two problems may arise if the algorithm works in the image space. The path planning procedure may create incorrect paths Connolly and Grupen 1993 and learning of appropriate control command vectors may also fail. Thus in this case it is necessary to represent the inverse kinematics set-mapping of the plant that maps workspace points to configuration space sets Locano-Pèrez and Wesley 1979 and one must use a configuration space representation.

Such mappings can be learnt in a self-organized manner Widrow, McCool, and Medoff 1978; Grossberg and Kuperstein 1986; Kawato, Furukawa, and Suzuki 1987; Miller 1987; Psaltis, Sideris, and Yamamura 1988; Mel 1988; Ritter, Martinetz, and Schulten 1988.

3.3.2 Controlling higher order plants

Speed field design for higher order plants may become complicated. This can be demonstrated by rewriting the dynamics of a higher order plant in the form of a first order differential equation whereupon one may arrive at a singular matrix field, $\mathbf{A}(\mathbf{q})$. Consequently the inverse field of $\mathbf{A}(\mathbf{q})$ is non-unique (the dimension of the control space may be smaller than that of the "phase-space".) This means that not all speed fields can be tracked with zero error; the speed field to be tracked should be carefully designed.

One method that solves this problem is best illustrated by a second order plant for which only $\ddot{\mathbf{q}}$ is directly controllable (as in the case of a robotic manipulator), i.e., when the plant's equation is given by

$$\ddot{\mathbf{q}} = \mathbf{A}(\mathbf{q}, \dot{\mathbf{q}})\mathbf{u} + \mathbf{b}(\mathbf{q}).$$

Assume that we have designed a speed field – still in the state space. Let us denote it by $\mathbf{v} = \mathbf{v}(\mathbf{q})$. Assume further that the intended equation of motion is given by $\dot{\mathbf{q}} = \mathbf{v}(\mathbf{q})$. Differentiating this equation along each trajectory leads to

$$\ddot{\mathbf{q}} = \mathbf{v}'(\mathbf{q})\mathbf{v}(\mathbf{q}), \tag{24}$$

where the prime denotes differentiation w.r.t. $\mathbf{q}$. Thus instead of following the speed field $\mathbf{v}(\mathbf{q})$, the plant should follow the acceleration field $\mathbf{v}'(\mathbf{q})\mathbf{v}(\mathbf{q})$. It follows that if $\dot{\mathbf{q}}(0) = \mathbf{v}(\mathbf{q}(0))$ then $\dot{\mathbf{q}}(t) = \mathbf{v}(\mathbf{q}(t))$ holds for all $t > 0$. However, the solution of (24) can be very sensitive to the inital condition. Even for linear plants the violation of the inital condition can result to an error exponentionally increasing with time. Thus this scheme must be used with care. Another possibility, which seems to be more robust, is that of using the inverse dynamics in a regulator position. Results concerning this method will be presented elsewhere.

3.3.3 Scaling issues

One of the most important questions that can arise in connection with algorithms is how they scale by size. For the present model the basic question in relation to scaling is the number of neurons and connections needed for the SA procedure. The estimation of the number of discretizing neurons necessary to achieve a given performance depends on the non-linearity of the control Equation (1) and is discussed later.

The number of the discretizing neurons is an exponential function of the dimensionality of the space they discretize. The exponential growth strongly limits the available fineness of the discretization. Let us assume a robot arm with six joints. If we want a tenfold discretization of every joint then the number of neurons we already need is 10^6. Having a fully connected, recurrent system the number of connections is 10^{12}. This problem is efficiently simplified by the utilization of the SA method.

The first advantage of the SA that used the geometry of the state space is that the number of neighbouring connections does not show the usual quadratic growth with the number of discretizing neurons, but grows in a linear fashion (for fixed state space dimension). The same holds for the motion connections, provided the number of motor neurons is fixed. Thus the full system is "linearly connected", which is a very attractive property. This property saves valuable digits in the exponent. Secondly, the SA the model provides results beyond the fineness of the discretization saving digits in the base.

Another important question is the time consumption of learning. The learning of different interneuron-control assocations are independent of each other. Having acquired the local movements at every point of the state space allows global path planning immediatly, since the local interneuron-motion associations link themselves into a whole control sequence. This feature promotes fast learning Benveniste, Métivier, and Priouret 1990. Conversely, to learn an interneuron-motion association may require a considerably long series of similar examples. If the control equation is highly non-linear, i.e. the r.h.s of Equation (1) changes rapidly with $\mathbf{q}$ and the examples are selected at random the learning might require a long time. A more elaborate solution would be to store information about the precision of control and apply reinforced exploration Scott and Markovich 1989.

3.3.4 Learning of non-linear dynamics

Other experiments show that the precision of control decreases when increasing the non-linearity of the plant's dynamics Szepesvári and Lőrincz 1995. In fact, dynamics (23) was used with different $\alpha_{\max}$ values and a fixed discretization. We measured the deviation of the motion vectors from the optimal ones determined through (18) and found that the average deviation increased linearly with increasing $\alpha_{\max}$. Based on this observation for a given precision and known non-linearity the number of discretization neurons can be approximately determined.

3.3.5 Non-linear activation spreading

In another set of experiments we considered the activation spreading mechanism of Glasius, Komoda, and Gielen 1994 called the continuous Dijsktra algorithm. We found that when this SA model is used the precision of learning decreases significantly. This is because in this model the activity map depends only on the position of the target and does not depend on the position of the plant while the activity map decays exponentially with the distance from the target. Thus far from the target the directional derivatives are approximately equal and this makes learning harder.

3.3.6 The effect of discretization

We have also investigated the effect of various state space discretizations. We have found that self-organized discretization of a two dimensional state space forms a hexagonal close packing and this is advantegeous to prewired grid-like discretizations. In particular, we found that the trajectories are smoother when the discretization is self-organized due to the close packing.

4 Conclusions

Complex control problems in structured environments were considered in the present work: the task being to control a plant with previously unknown dynamics while avoiding obstacles and experiencing perturbations in the plants dynamics. The solution is based on the designation of an appropriate speed field, which when tracked ensures collision free motion. This approach is

more robust than the earlier models for collision free motion, namely, trajectory tracking. A well known spreading activation neural model for fast speed field planning on the disretization of the state space was utilized. This model was augmented by interneurons and control neurons, with interneuron being connected to control neurons. The particular architecture allows to control plants having non-linear inverse dynamics. The resulting control signal is smooth.

5 Acknowledgments

We are grateful to Prof. András Krámli for his invalueable comments and suggestions. This work was partially founded by OTKA grants T017110, T014330, T014566, and US-Hungarian Joint Fund Grant 168/91-A 519/95-A.

6 Figure captions

Figure 1. The architecture of the network The discretizing neurons have spatially tuned filters. The neighbouring (or geometrical) connections connect discretizing neurons that represent neighbouring discretization points. Neighbouring connections are utilized for spreading activation. Interneurons perform associative learning with the control neurons.

Figure 2. Initial activities Start activities are negative and target activities are positive. The target and the plant are placed in the opposite corners.

Figure 3. Relaxed neural activity field Relaxed activities of neurons are shown for the external flow shown in Figure 2.

Figure 4. Trajectories for different control problems The left hand side and right hand side figures correspond to linear dynamics, the middle figure corresponds to the non-linear inverse dynamics given by (23). The right hand side figure shows that the neurocontroller generates collision free motions up to the precision of discretization.

Figure 5. Control signals by channel vs. time The figure corresponds to learnt control vectors and non-linear plant dynamics given by (23).

References

Amari, S. (1967). Theory of adaptive pattern classifiers. *IEEE Trans. Elect. Comput. 16*, 299–307.

Ben-Israel, A. and T. Greville (1974). *Generalized Inverses: Theory and Applications*. Pure and Applied Mathematics, Wiley-Interscience. New York: J. Wiley & Sons.

Benveniste, A., M. Métivier, and P. Priouret (1990). *Adaptive Algorithms and Stochastic Approximations*. Springer Verlag, New York.

Connolly, C. and R. Grupen (1993). On the application of harmonic functions to robotics. *Journal of Robotic Systems 10*(7), 931–946.

Dean, T. and M. Wellman (1991). *Planning and Control*. Morgan Kaufmann, San Mateo, CA, USA.

Fomin, T., C. Szepesvári, and A. Lőrincz (1994). Self-organizing neurocontrol. In *Proc. of IEEE WCCI ICNN'94*, Volume 5, Orlando, Florida, pp. 2777–2780. IEEE Inc.

Glasius, R., A. Komoda, and S. Gielen (1994). Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*.

Grossberg, S. and M. Kuperstein (1986). *Neural Dynamics of Adaptive Sensory-motor Control: Ballistic Eye Movements*. Elsevier, Amsterdam.

Hwang, Y. and N. Ahuja (1992). Gross motion planning – a survey. *ACM Computing Surveys 24*(3), 219–291.

Isidori, A. (1989). *Nonlinear Control Systems*. Springer-Verlag, Berlin.

Jordan, M. (1990). Learning and the degrees of freedom problem. In *Attention and Performance, XIII.*, Hillsdale. NJ: Erlbaum.

Kawato, M., K. Furukawa, and R. Suzuki (1987). A hierarchical neural-network model for control and learning of voluntary movements. *Biological Cybernetics 57*, 169–185.

Keymeulen, D. and J. Decuyper (1992). On the self-organizing properties of topological maps. In *Toward a Practice of Autonomous Systems, Proc. of the First European Conf. on Artificial Life*, pp. 64–69. MIT Press.

Lei, G. (1990). A neural model with fluid properties for solving labyrinthian puzzle. *Biological Cybernetics 64*(1), 61–67.

Lewis, F., C. Abdallah, and D. Dawson (1993). *Control of Robot Manipulators*. New York: MacMillan.

Locano-Pèrez, T. and M. Wesley (1979). An algorithm for planning collision-free paths among polyhedral objects. *Communications of ACM 22*(10), 560–570.

Lovelock, D. and H. Rund (1975). *Tensors, differential forms, and variational principles*. Pure and Applied Mathematics. A Wiley-Interscience Series of Texts, Monographs, and Tracts. Wiley-Interscience, New York.

Marshall, G. and L. Tarassenko (1994). Robot path planning using vlsi resistive grids. In *IEEE Proc., Vision, Image and Signal Processing*, Volume 141, pp. 267–272.

Martinetz, T. (1993). Competitive Hebbian learning rule forms perfectly topology preserving maps. In *Proc. of ICANN'93*, Amsterdam, The Netherlands, pp. 427–434. Springer-Verlag, London.

Mel, B. (1988). Murphy: A robot that learns by doing. In *Neural Information Processing Systems*, pp. 544–553. New York: American Institute of Physics.

Miller, W. (1987). Sensor based control of robotic manipulators using a general learning algorithm. *IEEE Journal of Robotics and Automation 3*, 157–165.

Miller, W. I., R. Sutton, and P. Werbos (Eds.) (1990). *Neural Networks for Control*. MIT Press, Cambridge, Massachusetts.

Miyamoto, H., M. Kawato, T. Setoyama, and R. Suzuki (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. *Neural Networks 1*, 251–265.

Morasso, P., V. Sanguineti, and T. Tsuji (1993). Neural network architecture for robot planning. In *Proc. of ICANN'93*, Amsterdam, The Netherlands, pp. 256–261. Springer-Verlag, London.

Narendra, K. and K. Parthasarathy (1990). Identification and control of dynamical systems using neural networks. *IEEE Trans. Neural Networks 1*(1), 4–27.

Psaltis, D., A. Sideris, and A. Yamamura (1988). A multilayered neural network controller. *IEEE Control Systems Magazine 8*, 17–21.

Ritter, H., T. Martinetz, and K. Schulten (1988). Topology conserving maps for learning visuo-motor coordination. *Neural Networks 2*, 159–168.

Robbins, H. and S. Monro (1951). A stochastic approximation method. *Ann. Mat. Stat. 22*, 400–407.

Rumelhart, D., G. Hinton, and R. Williams (1986). Distributed representations. In *Parallel Distibuted Processing: Explorations in the Microstructure of Cognition, vol.1: Foundations*. MIT Press, Cambridge, Massachuttes.

Scott, P. and S. Markovich (1989). Learning novel domains through curiosity and conjecture. In *Proceeding of the eleventh IJCAI*, pp. 669–674. Detroit, MI.

Szepesvári, C., L. Baláz, and A. Lőrincz (1994). Topology learning solved by extended objects: a neural network model. *Neural Computation 6*(3), 441–458.

Szepesvári, C. and A. Lőrincz (1995). Integrated architecture for motion control and path planning. *Journal of Robotic Systems*. submitted.

Szepesvári, C. and A. Lőrincz (1996). Approximate geometry representation and sensory fusion. *Neurocomputing*. (in press).

Tarassenko, L. and A. Blake (1991, April). Analogue computation of collision-free paths. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*, pp. 500–505. IEEE.

Vemuri, V. (1993). Artificial neural networks in control applications. *Advances in Computers 36*, 203–254.

Wasan, T. (1969). *Stochastic Approximation*. Cambridge University Press, London.

Werbos, P. (1988). Generalization of back propagation with applications to a recurrent gas market model. *Neural Networks 1*, 339–356.

Widrow, B. (1986). Adaptive inverse control. In *Proc. of the Second IFAC Workshop on Adaptive Systems in Control and Signal Processing*, Lund, Sweden, pp. 1–5. Lund Institute of Technology.

Widrow, B., J. McCool, and B. Medoff (1978). Adaptive control by inverse modeling. In *20th Asilomar Conference on Circuits, Systems and Computers*.

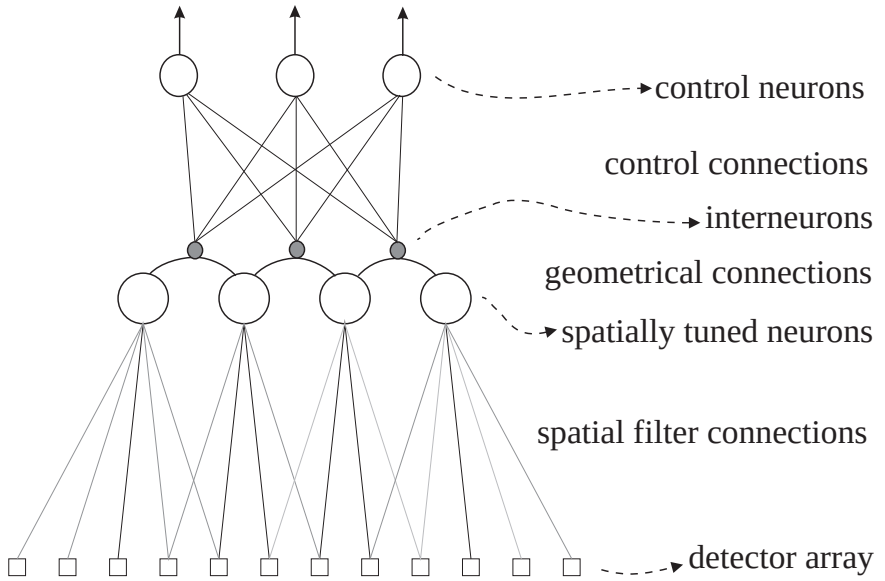

Figure 1: The architecture of the network.

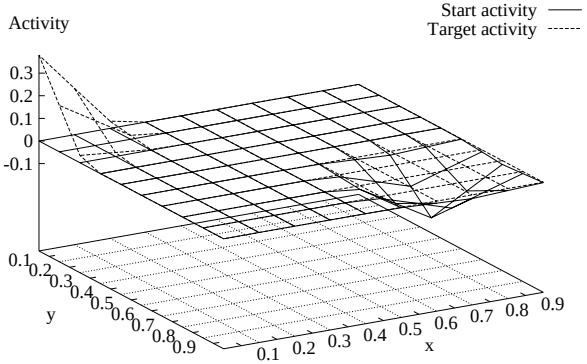

Figure 2: Initial activities

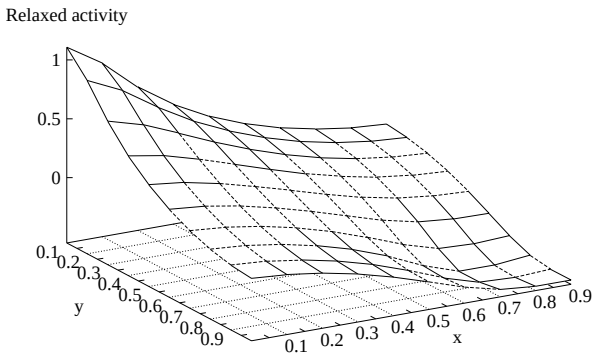

Figure 3: Relaxed neural activity field

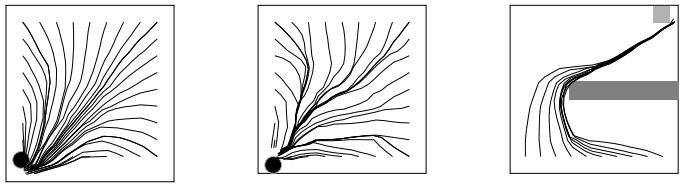

Figure 4: Trajectories for different control problems

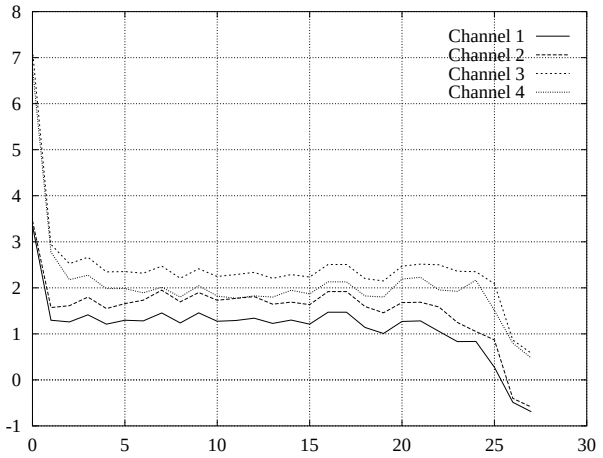

Figure 5: Control signals by channel vs. time