An integrated architecture for motion-control and path-planning

Csaba Szepesvári[‡] [†] and András Lőrincz[†] [*]

[†] 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, Aradi vrt. tere 1, Hungary, H-6720

[*] To whom correspondence should be addressed.

July 21, 1997

Abstract

We consider the problem of learning how to control a plant with non-linear control characteristics and solving the path-planning problem at the same time. The solution is based on a path-planning model that designates a speed field to be tracked, the speed field being the gradient of the equilibrium solution of a diffusion-like process which is simulated on an artificial neural network by spreading activation. The relaxed diffusion field serves as the input to the interneurons which detect the strength of activity flow in between neighboring discretizing neurons. These neurons then emit the control signals to control neurons which are linear elements. The interneuron to control-neuron connections are trained by a variant of Hebb's rule during control. The proposed method, whose most attractive feature is that it integrates reactive path-planning and continuous motion control in a natural fashion, can be used for learning redundant control problems.

Contents

1 Introduction **4**

2 Preliminaries **6**

2.1 Numerical solution: the ANN formulation 7

3 Architecture and functioning **8**

3.1 Coarse coding and gradient estimation 8

3.2 Following the gradient 10

3.3 The extended architecture: motion-control 11

3.4 Learning 12

4 Computational results **13**

5 Discussion **16**

6 Conclusions **18**

A Estimating the gradient by directional derivatives **20**

B Figure Captions **20**

List of Figures

1 The architecture of the network. 26

2 Initial activities 27

3 A typical relaxed neural activity field 28

4 Speed vector fields 29

5 Distance-to-go and components of the speed-vector vs. time 30

6 Components of control signals vs. time 31

7 Learnt control command vectors for linear and non-linear plants. 31

8 Learning curves vs. time 32

9 Error vs. non-linearity 33

1 Introduction

The subject of the paper is the learning of motion-control by an artificial neural network. Different aspects of the motion-control problem have been dealt with in the literature, including motion planning and motion execution among fixed or moving obstacles, known or (previously) unknown state-space structures, and motion-control of objects with different – and sometimes unknown – dynamical properties. Comprehensive reviews of motion planning and motion-control are already available in the literature[1, 2].

Throughout the paper it will be assumed that the state-space of the plant is restricted to a bounded region of $\mathbf{R}^n$. Further we presume that the relation between control signals and motion of the plant is given by

$$\dot{\mathbf{q}} = g(\mathbf{q}) + f(\mathbf{q})\mathbf{u}, \tag{1}$$

where $\mathbf{q}$, $\dot{\mathbf{q}}$ are respectively the state variable and speed-vector of the plant, $\mathbf{u}$ is the control signal and $g : \mathbf{R}^n \to \mathbf{R}^n$, $f : \mathbf{R}^n \to \mathbf{R}^{n\times m}$ are arbitrary C^∞ functions. Vast tracts of the control literature deals with (piecewise linear) controllability questions regarding Eq. 1.[3] In the case of robot manipulator control the state-space can be, say, the phase space or configuration space of the manipulator. In the first case (when the state-space is the phase space of the manipulator) it is known that a piecewise linear control signal may be derived for any point-to-point control problem. If the state-space is the configuration space of a robot then Eq. 1 means that the actuators are strong enough (or equivalently that the motion is slow enough) for us to be able to neglect the robot's mass. Later in the paper we indicate how this assumption might be circumvented.

A *path-planning task* is defined by three constituents, namely the *free space* $F \subset \mathbf{R}^n$, the "*start* state" and the "*target* state". The problem then is to find a control signal as a function of time that drives the plant from the initial state to the target state while the plant's states are restricted to F. The complement of F is called the *obstacle space* or the *forbidden zone*. A path-planning task is said to be well-defined if it has a solution. The problem considered here is that of finding a generic computational scheme that solves any well defined path-planning task.

Recently Lei[4] and Glausius et al.[5] have shown that motion planning may be performed with the help of an artificial neural network (ANN) algorithm, namely a spreading activation (SA) type of algorithm. Lei's model is identical to the harmonic function approach of Connolly

& Grupen's[6]. The main assumptions behind these models are the following:

A1 A discretization of the configuration space is given.

A2 That nearest neighbor discretization points are connected by a resistive (discretized diffusive) network.

A3 A controller is given that can be employed either for point to point control[4, 5] or for setting the speed-vector of the plant at any point[6].

The motion planning and execution procedure are composed of the following 4 steps:

Step 1. Initialize the diffusive process by identifying start and target states along with forbidden zones (obstacles) on the discretization.

Step 2. Start and relax spreading activation.

Step 3. Follow the gradient of the activity map at the actual state using the given controller.

Step 4. Repeat steps 1–3 until the goal state is reached.

The procedure is highly reactive and capable of collision free path-planning in stationary (and also non-stationary) environments provided that the representation of the environment is updated sufficiently often.[1] It can be shown that under appropriate conditions[2] the identified path is always close to the optimal path. Moreover, the bounded region between the path and the optimal path is a simply connected region of the free space.

In this work we extend previous SA studies by (1) proposing the use of coarse coding for representations, and (2) completing the model with control neurons that provide control signals. Coarse coding is implemented by introducing overlapping spatially tuned receptive fields (this can be developed by either a supervising or a self-organizing process [7, 8, 9, 10]) for the discretizing units. An important feature of the proposed controller is that it fits assumptions A1–A2 of the SA path-planning algorithms, it being capable of tracking any speed field as it implements a local approximation to the inverse-dynamics of the plant. Moreover, since the controller takes the form of a neural network, its working mechanism is highly parallel. Accompanying this, the control "knowledge" is stored in a distributed form over the weights of

[1] If the environment is static then a step-by-step recalculation of the activity map is unnecessary.

[2] The condition should be that at any point the minimal flow strength along the shortest path cannot be smaller than the flow strengths along suboptimal paths.

a network. Preliminary results relating to these issues were published previously[11] and indeed demonstrate the feasibility of this approach.

In the paper a computational example is provided that serves to illustrate certain points. Here a problem with a simple exact solution was chosen for the opportunity of inspecting the model's performance. But while the problem considered seems rather simple the case includes redundant control that is hard for a machine to learn[12].

The paper is organized in the following way. The next section offers a short summary of the path-planning model utilized here and the SA method that is used to implement the path-planning model. Section 3 describes the architecture and functioning of the proposed ANN. Pertinent numerical results are afterwards presented in Section 4, then Section 5 raises points about the learning of the dynamics of the controlled system and the scaling properties of the architecture. Finally some inferences drawn are enumerated upon in the closing section.

2 Preliminaries

In this section we review the works of Lei[4], Keymeulen and Decuyper[13], Connolly and Grupen[6], Glasius et al.[5] and Marshall and Tarassenko[14] that served as the theoretical basis for our algorithm. The neural SA method we consider can be viewed as the discretization of the following diffusion like differential equation:

$$\dot{\phi} = \Delta\phi + I, \tag{2}$$

where $\phi = \phi(\mathbf{q}, t)$ is the activity at a point $\mathbf{q}$ and time t, $I = I(\mathbf{q})$ 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 equilibrium solution of Eq. 2 by $\phi^* = \phi^*_{\mathbf{q}}$. Then the equation of motion of the plant is given by

$$\dot{\mathbf{q}} = \kappa\nabla\phi^*(\mathbf{q}), \tag{3}$$

where κ is a positive constant and $\nabla = (\partial/\partial x_1, \partial/\partial x_2, \ldots, \partial/\partial x_n)^T$ is the gradient (or 'nabla') operator.

The external flow is determined by the actual plant-state taking the value of 1 at the actual state and -1 at the target state, it being zero otherwise. In this way activity flows from the actual state towards the target state. The boundary conditions of Eq. 2 may be chosen to ensure collision-free paths either by the constraint $\frac{\partial \phi}{\partial \mathbf{n}}\Big|_{\partial F} = 0$ (Neumann boundary condition) or by $\phi\Big|_{\partial F} = c$ (Dirichlet boundary condition), or a combination of the two.

2.1 Numerical solution: the ANN formulation

Discretization points or *discretizing neurons* are evenly distributed in the state-space as the points of a mesh. Every discretizing neuron i has an associated position $\mathbf{c}_i$ in the state space and the response of a neuron to a state-space vector depends on its position, each neuron being spatially tuned. With the aid of the discretizing neurons the path-planning problem may be dealt with by distinguishing four types of neurons: *target neurons* corresponding to the target state, *start neurons* corresponding to the start state, *active neurons* corresponding to the free-space, and *inactive neurons* corresponding to the forbidden region of the state space. It is furthermore assumed that there is only one target and one start neuron and that the set of active and inactive neurons is disjoint. Then the degree to which every neuron participates in any of the above classes is either 0 or 1.

The diffusion process (Eq. 2) is simulated on the discretizing layer by the following *recurrent* equation:

$$\dot{\sigma_i} = I_i + \sum_{k \in N_i \cap F} (\sigma_k - \sigma_i), \quad i \in F \tag{4}$$

where F is the set of active neurons (corresponding to the free space) and N_i denotes the set of neighboring neurons at neuron i, and σ_i is the discretized version of the activity flow ϕ at position $\mathbf{c}_i$. The corresponding external signal I_i has a value of 0, 1 or -1, that is $I_i = 1$ if neuron i is the start neuron, $I_i = -1$ if neuron i is the target neuron, and $I_i = 0$, otherwise. Because neurons lying outside the free-space are not involved in the computation activation avoids obstacle regions.

The subsequent state of the plant is associated with a neuron representing the neighboring position of the start neuron with the steepest activity drop. Afterwards the plant is moved to the new state the procedure is repeated. Namely, the path-planning task is transformed onto the neuronal layer, spreading activation (SA) starts and settles down and the next state is determined according to the "gradient" of the equilibrium activation.

3 Architecture and functioning

In this section we introduce the architecture and 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 (Eq. 1) and path generation equation (Eq. 3) are given. This combination motivates an extension of the architecture with interneurons, control neurons and control command connections, which is described next. Finally, it is demonstrated how associative direct inverse learning can be applied to estimate the optimal control command vectors.

3.1 Coarse coding and gradient estimation

Smooth trajectories are desirable in many applications. In the ANN model previously elaborated upon there are two sources of non-smooth trajectories. They are

1. Discretizing neurons providing binary outputs.

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

In fact the second implies the first, and if it is relaxed then the first assumption yields an ambiguous (or inaccurate) problem representation. Luckily, this problem can be circumvented in the following way:

Firstly, the assumptions are relaxed by allowing the neurons to develop continuous response signals and enabling coarse coding at the same time. Our discretizing units provide a gracefully-overlapping degrading spatially-tuned representation that can be developed in a self-organizing fashion say[8, 9]. The spatially-tuned response of the discretizing units can then be used to determine the 'center of the receptive field' by means of weighted averaging. Regularization-based radial basis function networks[7] represent another tool that can be utilized for developing a coarse coded representation. In this way a fuzzy or coarse coded representation of the path-planning problem is obtained that is advantegous in many respects[15]. In Section 4 there is a figure illustrating this coarse coding. As a consequence the "fuzzy set" of start, target, active and inactive neurons may actually overlap. For reasons of prudence we classify neurons having above-threshold obstacle activities as inactive neurons, while other neurons are classed as active.

Secondly, since start and target neurons are no longer unique the external flow I_i will be determined by $I_i = s_i/s - t_i/t$ 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. The diffusion of activity is left unchanged.

Thirdly, since the plant's state is now coarse coded we now have to reconsider the implementation of the equation of motion (Eq. 3). Firstly, the gradient of the equilibrium flow $\phi^*(\mathbf{q})$ can be approximated by the sum of the relevant directional derivatives, the 'geometry vectors' of neighboring neurons providing the directions used in the approximation. The geometry vector between neuron i & j whose center positions are $\mathbf{c}_i$ & $\mathbf{c}_j$ is the vector that points from $\mathbf{c}_i$ to $\mathbf{c}_j$; that is $\mathbf{g}_{ij} = \mathbf{c}_j - \mathbf{c}_i$. Denoting the equilibrium activity at node i by σ_i the gradient of the equilibrium activity flow at node i is approximated by

$$\mathbf{d}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{g}_{ij}, \tag{5}$$

where

$$I_{ij} = (\sigma_j - \sigma_i) w_{ij} \tag{6}$$

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

$$w_{ij} = \frac{1}{||\mathbf{c}_j - \mathbf{c}_i||}, \quad j \in N_i. \tag{7}$$

The w_{ij} values are defined only for neighboring neurons and are called the strength of *neighboring connections* between neurons i and j. Since the plant-state is represented by the blob $\{s_i\}_i$, the gradient of the flow at $\mathbf{q}$ is approximated by the gradient-flow components $\{\mathbf{d}_i\}_i$ weighted by $\{s_i\}_i$:

$$\mathbf{d} = \sum_{i \in F} s_i \mathbf{d}_i, \tag{8}$$

Note that in Eq. 5 it is necessary to restrict the summation for active nodes (elements of F) since activities, and hence I_{ij} values, are only defined for active nodes. The I_{ij} values may be interpreted as the activity flow along the neighboring connection between neuron i and j. It is worth noting as well that the connection structure $\{I_{ij}\}$ can be developed by means of standard self-organizing procedures[16, 17, 18, 9, 10].

3.2 Following the gradient

In order to realize the planned speed-vector $\mathbf{d}$ (Eq. 8) the control equation of the plant (Eq. 1) must also be taken into account. Let us assume that the plant is invertible, i.e. there exists at least one $\xi : \mathbf{R}^n \times \mathbf{R}^n \rightarrow \mathbf{R}^m$ mapping[3] satisfying

$$\mathbf{d} = g(\mathbf{q}) + f(\mathbf{q})\xi(\mathbf{q}, \mathbf{d}), \tag{9}$$

where $\mathbf{d}, \mathbf{q} \in \mathbf{R}^n$ are arbitrary vectors. In other words $\xi(\mathbf{q}, \cdot)$ is the right inverse of the function $h(\mathbf{q}, \cdot) = g(\mathbf{q}) + f(\mathbf{q})\,\cdot$ for every $\mathbf{q}$ which is called the inverse-dynamics of the plant. Using the inverse-dynamics the static state-feedback control of the plant can be written as

$$\mathbf{u} = \xi(\mathbf{q}, \kappa\nabla\phi(\mathbf{q})), \tag{10}$$

where $\mathbf{q}$ is the plant-state.

In order to use Eq. 10 in practice we have to represent ξ in a suitable way. First let us fix an arbitrary point $\mathbf{q}$ in the state-space. This point may be chosen as some discretization point, $\mathbf{c}_l$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathbf{R}^n$ denote k "direction" vectors $(k \geq n)$, which one might think of as geometrical vectors associated with a particular discretization point. Furthermore, suppose that the contol vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k \in \mathbf{R}^m$ satisfy the equalities

$$\mathbf{v}_i = g(\mathbf{q}) + f(\mathbf{q})\mathbf{u}_i, \; i = 1, \ldots, k. \tag{11}$$

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

$$\mathbf{d} = \sum_{i=1}^{k} \alpha_i \mathbf{v}_i, \quad \text{with} \sum_{i=1}^{k} \alpha_i = 1. \tag{12}$$

Note that if there are at least $n+1$ vectors among the vectors $\mathbf{v}_i$ that are affine independent (i.e. every n of those vectors span $\mathbf{R}^n$), then coefficients that satisfy $\sum_i \alpha_i = 1$ can always be

[3] There may be more than one such mapping. In the degenerate case one can always find an arbitrary, suitable (smooth up to some desired degree, etc.) mapping.

found. Let us therefore consider the control vector

$$\mathbf{u} = \sum_{i=1}^{k} \alpha_i \mathbf{u}_i. \tag{13}$$

Substituting Eq. 13 into Eq. 1 we have that the control vector $\mathbf{u}$ yields the speed-vector $\mathbf{d}$, meaning that a local approximation of the inverse-dynamics function is feasible in the implicit form $\{(\mathbf{v}_i, \mathbf{u}_i)\}$.

3.3 The extended architecture: motion-control

Assume that we are given a particular path-planning problem and the recurrent network has already relaxed into a stationary state. The plant's speed-vector must then correspond to the gradient of the equilibrium flow in the starting state so we are permitted to use the approximation of the gradient in the starting state developed in Section 3.1 given by $\mathbf{d} = \sum_{i \in F} s_i \mathbf{d}_i$, where $\mathbf{d}_i$ is the approximated gradient at neuron i, that is $\mathbf{d}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{g}_{ij}$, where $I_{ij} = w_{ij}(\sigma_j - \sigma_i)$. According to the previous section if the control vectors $\mathbf{u}_{ij}$ are given ($j \in N_i$) and satisfy

$$\mathbf{g}_{ij} = g(\mathbf{c}_i) + f(\mathbf{c}_i)\mathbf{u}_{ij} \tag{14}$$

then

$$\mathbf{u}_i = \sum_{j \in N_i \cap F} I_{ij} \mathbf{u}_{ij} \tag{15}$$

moves the plant into the direction $\mathbf{d}_i$ provided that the plant is in state $\mathbf{c}_i$.[4] Taking into account the coarse coding of the plant-state, i.e. that the state of the plant is given by the blob of activities $\{\, s_i \,\}$ we get that the vector

$$\mathbf{u} = \sum_i s_i \mathbf{u}_i \tag{16}$$

which can be used to move the plant along the approximate gradient $\mathbf{d}$.

The computation in Eq. 15 and 16 fit in well with the recurrent architecture that computes the equilibrium flow if we extend the architecture with control neurons and interneurons. Control neurons provide control signals and are connected to interneurons via command con-

[4]In Eq. 15 the I_{ij} coefficients may be normalized but according to our numerical experiments Eq. 15 can work equally well.

nections. Interneurons in contrast correspond to neighboring connections and monitor the activities that flow along the connections and provide proportional outputs with then, so 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 at control neuron k be the kth component of $\mathbf{u}_{ij}$. Then the motion planning and execution procedure is just

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

Step 2. Compute the equilibrium flow by activation spreading.

Step 3. Compute the output of interneurons, i.e. the directional derivatives of the flow weighted by the coarse coding activities of the actual plant-state.

Step 4. Compute the control signal of each control neuron as a weighted sum of interneuron outputs, where the weights are those of the command-connections.

The corresponding architecture for this is shown in Fig. 1.

3.4 Learning

In this section we will show how control vectors satisfying Eq. 15 can be learnt by the neural net, the proposed learning scheme being similar to direct inverse modeling[19, 20, 21]: The control signal and observed movement produced by the plant in response to the control signal provide the input for learning. However, our method is not simply another form of error back-propagation (or in control theoretical terms not just the method of variations). The main point of the algorithm is that observed movements of the plant are represented by an equilibrium-flow corresponding to the specific external flow I_i, when the source is the coarse-coded representation of the initial plant-state and the sink is the same for the state of the plant after the observed movement. In this way the algorithm is fully self-organized and self-contained. The steps of the algorithm are as follows:

Step 1. Develop a 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 activity vector.

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

Step 5. Associate the control signal with interneurons weighted by interneuron-outputs.

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

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

where $0 < \alpha_{ij} < 1$ is the learning rate of interneuron (i, j). The learning rate can be either time dependent or stationary. In principle it is reasonable to choose a Robbins-Monro type of time dependence[22, 23] to ensure the convergence of $\mathbf{u}_{ij}$ to the time-averaged values of the learning samples and also to retain adaptivity. When using such decaying learning rates the adaption-rate may become extremely low over time. However, if the learning rate is kept constant the adaptivity may be kept above a certain predefined level. In such a case $\mathbf{u}_{ij}$ will be a stochastic variable with mean given by the sample average and deviation magnitude asymptotically proportional to $\sqrt{\alpha_{ij}}$ [24].

It can be shown that the sample-average satisfies Eq. 15 if the sampling is complete and if equilibrium flows satisfy some special conditions[25]. Another possible learning method would be to store the "best-command-vector-so-far". However, while this method may be sensitive to state errors and noise, Hebbian-learning filters out noise without any difficulty.

4 Computational results

In the examples subsequently presented the state-space is the two dimensional rectangle in $[0, 1] \times [0, 1]$, while the control system of the plant has four components. This means that we are treating a redundant control problem, the degrees of freedom in the motor command space being higher than the degrees of freedom in the task space. As pointed out by Jordan such redundancy cannot be solved by error back-propagation: there are an infinite possible permutations of command errors that lead to the same error expressed in task coordinates[12]. But as will be seen and should already be clear from the previous discussion our system is quite capable of dealing with redundant control tasks.

The control problem outlined here is a simple type of *sensory-motor control.* A simulated camera, that is a pixel discretization camera, provides the input for the system and the discretizing neurons work on the "image space" of the simulated camera instead of the state-space.

However the "product" of the two discretizations, the discretization generated by the camera (as a function of the state to the image space) and the discretization generated by the neurons (as a function mapping the image space to neuronal activities) is itself a discretization of the state space. Since neighboring connections reflect neighboring relationships in the state-space – i.e. the discretization is topographic[10] – the extra discretization step does not affect the working of the model apart from an effect on the fineness of the product discretization.

Two control problems are now cited for comparison: one with a linear and another with non-linear dependence on $\mathbf{q}$. In the linear case the control components alone move the plant respectively towards north, east, south and west. The control equation of the plant is simply 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:

$$f(\mathbf{q}) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} f_0, \tag{17}$$

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 all the experiments except one when the effect of $\alpha_{\max}$ on learning was measured the setting $\alpha_{\max} = \frac{\pi}{2}$ was employed. According to Eq. 17, the speed-vector of motion is the rotated speed-vector of the linear case within the circle centered on the point $(0.5, 0.5)$ and having a radius of 0.5. The rotation angle is state dependent – it is $\pi/2$ at the centre point $(0.5, 0.5)$ and decreases with distance from the center. Outside the circle the rotation is zero.

The purpose of the trials was to demonstrate

(i) completeness, i.e. the model is capable of path-planning and *execution* in the case of non-trivial, non-linear control problems, and

(ii) learning capabilities, i.e. to compare the typical results of learned and "perfectly" preprogrammed (prewired) models.

The initial activities are shown in Fig. 2, the start activities being negative and target activities being positive, while the equilibrium field generated by the diffusion process is depicted in Fig. 3. The target and plant were placed in opposite corners. Clearly, the initial activities shown in Fig. 2 provide a coarse-coded representation of the start and target configurations.

The results of the learning experiments are presented in two ways, as speed-vector fields tracked by the plant and as speed-vectors corresponding to the control commands of individual interneurons. Fig. 4 shows speed-vector fields for the two different cases. In both the target is always at the lower left corner of the state-space, whereas the plant's position varies over the whole state-space. A speed-vector of the figure represents the speed of the plant when the plant is placed at the start-position of the vector. Thus paths followed by the plant ought to be tangential to the depicted speed-vector field (the speed-vector field is always tangential to the plant path).[5] The left and right columns of Fig. 4 correspond to "perfectly" prewired (in the sense that defined by Eq. 14) and learnt control commands respectively, while the upper and lower rows depict the linear and non-linear cases. At the edges of the state-space the speed-vectors point somewhat inside the region rather than towards the target. This property is the consequence of the applied linear gradient estimation and not the path-planning procedure, the Neumann type boundary condition having been used in these trials. The suggested procedure restricts the participating directions in such a way that the linear combination lacks an outwardly pointing component. Apart from this edge effect the speed-vectors point towards the target. Moreover from Fig. 4 it seems that only minor differences are detectable when comparing the prewired and learnt control commands cases. Notice too that as a result of coarse coding and a linear gradient estimation the speed-vector field is continuous, i.e. the trajectory of the plant should be "smooth".

The smoothness feature of Fig. 4 is readily observed in Fig.s 5 and 6 which show the components of the speed-vector, the path length, and components of the control vectors vs. time, respectively, when the plant is proceeding from one corner to the opposite one. The corresponding figures in the conventional discretization scheme should show discrete time jumps. But as was noted above the (relative) smoothness of the curves is a result of the consistent application of the coarse coding technique. The residual 'noise' in the signals can be further diminished by enlarging the number of discretizing units (which will also decrease the struc-

[5] Complex path-planning tasks are not introduced here since previous studies[4, 13, 6, 5] have already demonstrated the potential of the method as a viable path-planning procedure.

tural approximation error of the architecture) or by increasing the sampling rate at which the equation of motion was simulated. Both figures correspond to learnt control vectors and of the non-linear plant.

A fairly accurate view of the learning capabilities of the model is given by Fig. 7, which illustrates those speed-vectors corresponding to the control commands of individual interneurons. The small circles in the figure correspond to neurons (discretization points), while connections between neighboring neurons have been left out. Two interneurons and thus two control commands are associated with each connection. The first control command moves the plant from the one discretization neuron to the other, while the other control command moves the plant in reverse, from the second discretization neuron to the first. As control commands are 4 dimensional they cannot be easily shown on paper. Instead both speed-vectors that result from executing those commands which are at the starting discretization neurons are shown in the Figure. Speed vectors should be parallel to the corresponding connection in the ideal case. Speed vectors could also be learnt almost perfectly both for the linear (left) and non-linear (right) control problems. As can be seen the differences between the subfigures are pretty minor, the unexpected coincidences being due to the same set of learning examples having been applied. Further training would probably lead to a further improvement in the control command vectors, as may be inferred from Fig. 8 which shows the learning curves for different non-linearity ($\alpha_{\max}$ values) cases. However, there may be a theoretical limit of accuracy that depends on the growth rate of the non-linearity. After inspecting Fig. 9 the results for the speed-vector errors can be seen after learning under similar conditions has taken place, with identical training examples but different non-linearities ($\alpha_{\max}$ values). The linear growth with increasing $\alpha_{\max}$ indicates that to retain good precision with higher non-linearities both the training time and the fineness of discretization should be suitably increased.

5 Discussion

In the previous section we touched upon the question of sensory-motor control. In this situation data from a sensor-space serves as the input for the algorithm instead of the state-space. In the example given the sensor-space was a discretized version of the state space. In the general case however the connection between the sensor-space and state-space is non-trivial. Consider, for instance, a robot manipulator with more than 3 degrees of freedom (k say) in a 3D space, and assume that two cameras monitor the end part of the manipulator. Then the state-space is k-dimensional while the workspace and image manifold are both 3 dimensional. Two problems may arise if the algorithm works in the image space. Firstly, the path-planning procedure could create incorrect paths[6] and the learning of appropriate control command vectors might also fail. In such a case it would be necessary to represent the inverse kinematics set-mapping of the plant that maps workspace points to configuration space sets[26], and one would also have to use a configuration space representation. Happily, such mappings can be learnt in a self-organized manner.[19, 21, 27, 28, 20, 29, 30]

One of the most important questions that can arise in connection with algorithms is how they scale up in size. For the present model the basic question in relation to scaling is that of the number of neurons and number of connections needed for the spreading activation (SA) procedure. The estimation of the number of discretizing neurons required to achieve a given performance depends on the stability properties of control Eq. 1 and is beyond the scope of this paper. However, the scaling properties of the model can be dealt with quite easily.

The number of the discretizing neurons is an exponential function of the dimensionality of the space they discretize. Evidently, the exponential growth strongly limits the available fineness of the discretization. Let us for the sake of argument assume a robot arm with six joints. If we would like a tenfold discretization of every joint then the number of neurons we already need is 10^6. With a fully connected, recurrent system the number of connections rises to 10^{12}. Fortunately this problem is considerably simplified with the aid of the SA method.

The first advantage of the SA approach is that the number of neighboring connections with the number of discretizing neurons does not show the usual quadratic growth, but rather grows in a linear fashion. The same holds for the control connections provided the number of control neurons is fixed. Hence the full system is "linearly connected", which is a very attractive property since it saves valuable digits in the exponent. Secondly, the SA model together with the consistent utilization of coarse coding provides results beyond the fineness of the discretization, saving digits in the base.

The problem of inertia (i.e. the dynamics) increases the demand for discretizing neurons: if the dynamics cannot be neglected the brute-force approach dictates that the state space be the phase space. In this case the dimension of the state-space is doubled and storage requirements are quadrupled. But the advantage of using the phase space is that it enables one to plan time- or energy-optimal trajectories.

In the ideal case when the approximation to the inverse-dynamics is accurate the aforementioned control method just cancels out the non-linearities of the plant. However, the estimate of the inverse-dynamics is usually imprecise and thus raises stability issues. In the examples given stability was achieved by having a special form of the speed fields to be tracked. Several methods exist to make the inverse-dynamics based control more robust. One such method is to use the inverse-dynamics both in a static state-feedback (like that in this paper) and a dynamic state-feedback position[31]. An extension with PD/PID controllers have also been proposed in the adaptive control and neural network literature[32, 33, 34]. Notice that inverse-dynamics based control is able to work with plants of any relative degree provided one starts from the normal form of the plant[3, 35].

A question of great importance is the time taken up in learning. Since the learning processes of different interneuron-control assocations are independent of each other, having acquired the local approximants at every point of the state-space, it allows for instant "global" motion-control since the local interneuron-motion associations link themselves into a whole control sequence. This feature can indeed promote fast learning.[36] Conversely, to learn an interneuron-motion association may require a considerable number of similar examples. If the control equation is highly non-linear, i.e. the r.h.s of Eq. 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[37].

6 Conclusions

We have presented an integrated control algorithm that is capable of controlling a plant with non-linear dynamics in the presence of obstacles. The control algorithm can work on-line and is highly reactive. The underlying path planning procedure is based on a diffusion field generated by the Laplace equation. The path-planning algorithm is known to be complete: if the representation of the environment is accurate and there exists a path from the start to the target then the algorithm is capable of finding it (in fact it is able to find a near optimal path) in contrast with some artificial potential-field approaches[1]. The path-planning procedure is global in its scope[1]: it needs a complete representation of the environment. In the case of incomplete information the algorithm may choose suboptimal paths. However, if the representation of the environment is updated on-line according to the incoming experiences then the algorithm will find the solution eventually[6, 5].

In the article the usual path-planning procedure was extended with local computations to solve the plant control problem. Although, the resulting algorithm was found to be similar to approximations based on radial-basis functions, we needed to extend the basic model by using localised computations to solve the rendering of directions to control commands. Associative learning was then introduced for finding the optimal-control command vectors. The learning algorithm is even capable of functioning in the case of redundant control when the dimension of the state-space is lower than the dimension of the control space. The learning of connections is efficient since the coarse coding of the discretization layer enables the training of several control-command vectors at the same time. The learning speed can be further raised by introducing neighbor training based on neighboring connections of the discretization layer. Furthermore, the generalization capabilities and learning speed of the network could be improved somewhat by introducing a feedback from the control performance to the discretization neurons. The neighboring discretization neurons with identical control command neighborhoods can be integrated while in regions where accuracy does not suffice new discretization neurons might be introduced.[38, 39, 40] Lastly, in numerical studies it was realised that plants with non-linear dynamics are readily capable of being controlled with our proposed approach.

Appendix

A Estimating the gradient by directional derivatives

Gradient estimation by means of directional derivatives lies at the heart of our control scheme. Hence a brief description of the underlying gradient estimation will now be given.

Consider two neighboring neurons, i and j and the equilibrium solution ϕ^* of Eq. 2. Let $\phi_i^* = \phi^*(\mathbf{c}_i)$. Then the directional derivative of ϕ^* at point $\mathbf{c}_i$ with respect to $\mathbf{g}_{ij}$ may be approximated as the activity difference:

$$\nabla_{\mathbf{g}_{ij}} \phi^* \approx \gamma_{ij}^* = \frac{\phi_j^* - \phi_i^*}{||\mathbf{c}_j - \mathbf{c}_i||}. \tag{18}$$

Similarly, the gradient of ϕ^* at the point $\mathbf{c}_i$ can be approximated by

$$\nabla \phi^* \Big|_{\mathbf{q}=\mathbf{c}_i} \approx \sum_j (\nabla_{\mathbf{g}_{ij}} \phi^*) \mathbf{g}_{ij} \tag{19}$$

which may be further approximated by substituting Eq. 18 into Eq. 19:

$$\nabla \phi^* \Big|_{\mathbf{q}=\mathbf{c}_i} \approx \sum_j \gamma_{ij}^* \mathbf{g}_{ij} \tag{20}$$

which is the final approximation result.

B Figure Captions

Figure 1 **The architecture of the network.** Traversing from the bottom to the top of the picture one can see four different layers of neurons, whose functions are readily apparent. The discretizing neurons have spatially-tuned filters that can be formed by tuning a radial basis function network or using a self-organizing competitive scheme. The geometrical connections that link discretizing neurons in the first layer represent neighboring relationships and can be developed by Hebbian-learning. As regards functionality, the start- and target-activities are created on the layer of discretizing neurons by some recognition modules not mentioned here, and these are relaxed via activation spreading through the geometrical connections. The interneurons sitting on the geometrical connections emit a signal proportional to their detected activity flow which go to control neurons. Besides this they also perform associative learning with control neurons that shapes the command connections. This associative learning forms the basis of approximating the inverse dynamics.

Figure 2 **Initial activities** As can be seen the start activities are negative whereas the target activities are positive. The target and the plant are placed in the opposite corners, the neuronal discretizing layer being a rectangular array for ease of visualization.

Figure 3 **A typical relaxed neural activity field** The relaxed equilibrium activities, corresponding to the external flow shown in Fig. 2, have been plotted here. The plant should move along the gradient of this equilibrium field.

Figure 4 **Speed vector fields** Upper row: linear plant, lower row: non-linear plant. Left column: prewired control command connections, right column: learnt control command connections. The arrows represent the plant-speed at different starting points when the target is placed in the lower left corner. The path of the plant started from any point is always tangential to the speed-vector field shown. This means that the plant moves to the target (lower left corner) in a more or less straight line. The magnitude of the arrows represent the magnitude of the plant speed-vector at different positions.

Figure 5 **Distance-to-go and components of the speed-vector vs. time** The figure shows the distance left to reach the target, the plant speed and the two components of the speed-vector of the plant as a function of time while the plant is moving from the upper right to the lower left corner of the unit square. Path-length values have been multiplied by 10 for normalization.

Figure 6 **Components of control signals vs. time** The figure shows the output of individual control neurons as a function of time while the plant is moving from the upper right to the lower left corner of the unit square.

Figure 7 **Learnt control command vectors for linear and non-linear plants.** The arrows start from interneurons situated between discretizing neurons. These represent the speed-vector of the plant provided the command vector of the corresponding interneuron is the control signal. As can be seen the differences between the two subfigures are quite minor.

The strange similarities between the two parts are actually due to the same set of learning examples being adopted; with another choice the similarities disappear.

Figure 8 **Learning curves vs. time** The curves depict the time-development of the error of the approximation to the inverse-dynamics for different types of non-linearities ($\alpha_{\max}$ values). The error is defined as the maximum deviation of the approximated inverse-dynamics from the corresponding theoretical value.

Figure 9 **Error vs. non-linearity** Average error of the approximation to the inverse-dynamics as a function of the non-linearity present in the inverse-dynamics after completing identical training courses. After an initial phase the curve shows a linear dependence with the rate of growth of the non-linearity of the inverse-dynamics.

References

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

2. E.D. Sontag. Some topics in neural networks and control. Technical report ls93-02, Department of Mathematics, Rutgers University, New Brunswick, NJ 08903, 1993.

3. A. Isidori. *Nonlinear Control Systems*. Springer-Verlag Berlin, Heidelberg, 1989.

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

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

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

7. T. Poggio and F. Girosi. Regularization algorithms for learning that are equivalent to multilayer networks. *Science*, 247:979–982, 1990.

8. T. Kohonen. *Self Organisation and Associative Memory*. Springer-Verlag, Berlin, 1984.

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

10. Cs. Szepesvári and A. Lőrincz. Approximate geometry representation and sensory fusion. *Neurocomputing*, 12(2–3):267–287, July 1996.

11. Cs. Szepesvári, T. Fomin, and A. Lőrincz. Self-organizing neurocontrol. In *Proceedings of ICANN'94*, pages 623–626, Sorrento, Italy, May 1994.

12. M.I. Jordan. Learning and the degrees of freedom problem. In M. Jeannerod, editor, *Attention and Performance, XIII.*, Hillsdale, 1990. NJ: Erlbaum.

13. D. Keymeulen and J. Decuyper. On the self-organizing properties of topological maps. In F.J. Varela and P. Bourgine, editors, *Toward a Practice of Autonomous Systems, Proc. of the First European Conf. on Artificial Life*, pages 64–69. MIT Press, 1992.

14. G.F. Marshall and L. Tarassenko. Robot path planning using vlsi resistive grids. volume 141, pages 267–272, 1994.

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

16. T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, M. Mäkisara, O. Simula, and J. Kangas, editors, *Proceedings of ICANN*, volume 1, pages 397–402. Elsevier Science Publishers B.V., Amsterdam, 1991.

17. Cs. Szepesvári, L. Balázs, and A. Lőrincz. Topology learning solved by extended objects: a neural network model. In S. Gielen and B. Kappen, editors, *Proc. of ICANN'93*, page 678, Amsterdam, The Netherlands, September 1993. Springer-Verlag, London.

18. T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

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

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

21. S. Grossberg and M. Kuperstein. *Neural Dynamics of adaptive sensory-motor control: Ballistic eye movements.* Amsterdam: Elsevier, 1986.

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

23. T. Wasan. *Stochastic Approximation.* Cambridge University Press, 1969.

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

25. Cs. Szepesvári and A. Lőrincz. On learning correct control commands in an integrated neurocontrol architecture. 1995.

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

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

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

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

30. H.J. Ritter, T. Martinetz, and K.J. Schulten. Topology conserving maps for learning visuomotor coordination. *Neural Networks*, 2:159–168, 1988.

31. Cs. Szepesvári, Sz. Cimmer, and A. Lőrincz. Dynamic state feedback neurocontroller for compensatory control. *Neural Networks*, 1997. (in press).

32. J. Craig, P. Hsu, and S. Sastry. Adaptive control of mechanical manipulators. *Int. J. of Robotic Research*, 6(2):16–28, 1987.

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

34. F.L. Lewis, K. Liu, and A. Yesildirek. Neural net robot controller with guaranteed tracking performance. *IEEE Trans. on Neural Networks*, 6(3):703–715, May 1995.

35. S. Sastry and M. Bodson. *Adaptive Control - Stability, Convergence and Robustness.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

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

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

38. B. Fritzke. Let it grow – self organizing feature maps with problem dependent cell structure. In T. Kohonen, M. Mäkisara, O. Simula, and J. Kangas, editors, *Proceedings of ICANN*, volume 1, pages 403–408. Elsevier Science Publishers B.V., Amsterdam, 1991.

39. P. van der Smagt, F. Groen, and F. van het Groenewoud. The locally linear nested network for robot manipulation. In *Proceedings of the IEEE Int. Conf. on Neural Networks*, pages 2787–2792, Orlando, Florida, May 1994. IEEE Press.

40. Sz. Kovács, G.J. Tóth, and A. Lőrincz. Output sensitive discretization for the help of genetic algorithm with migration. to be published elsewhere.

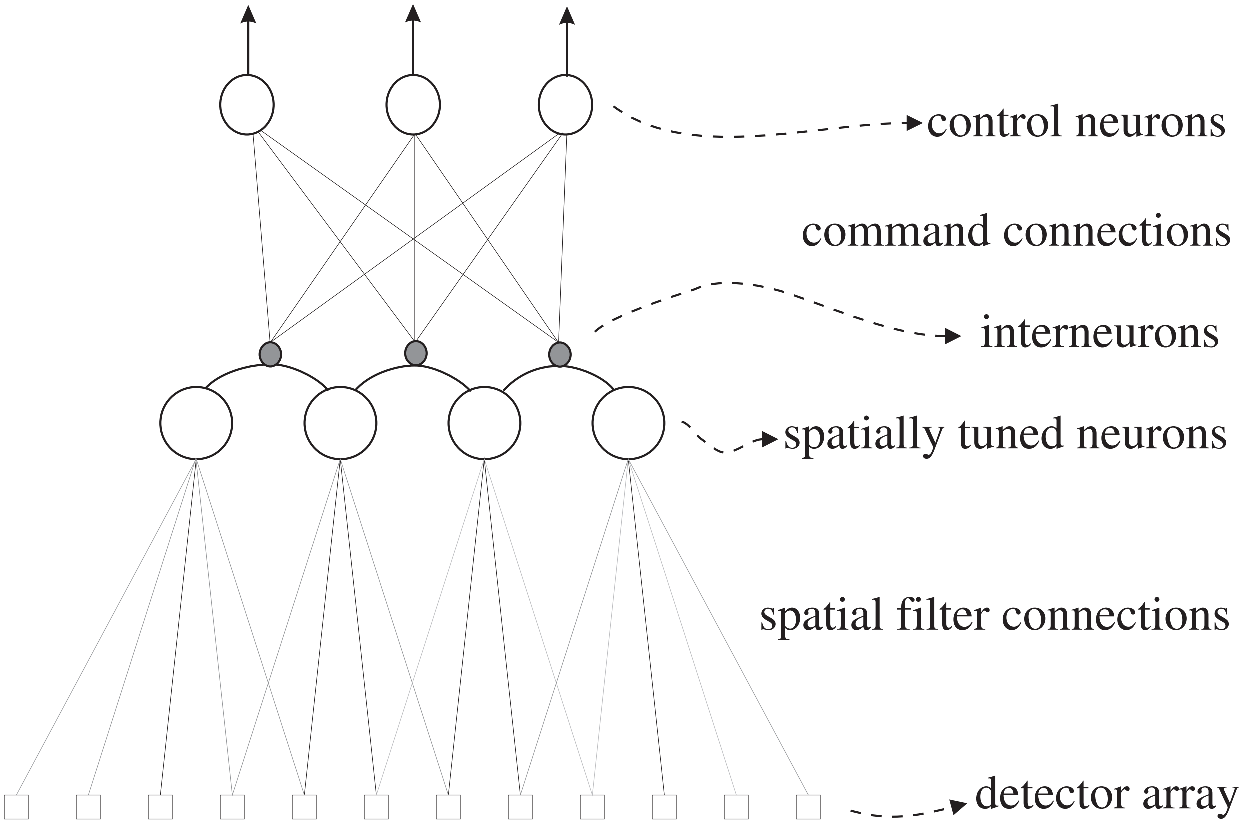

Figure 1: The architecture of the network.
Traversing from the bottom to the top of the picture one can see four different layers of neurons, whose functions are readily apparent. The discretizing neurons have spatially-tuned filters that can be formed by tuning a radial basis function network or using a self-organizing competitive scheme. The geometrical connections that link discretizing neurons in the first layer represent neighboring relationships and can be developed by Hebbian-learning. As regards functionality, the start- and target-activities are created on the layer of discretizing neurons by some recognition modules not mentioned here, and these are relaxed via activation spreading through the geometrical connections. The interneurons sitting on the geometrical connections emit a signal proportional to their detected activity flow which go to control neurons. Besides this they also perform associative learning with control neurons that shapes the command connections. This associative learning forms the basis of approximating the inverse dynamics.

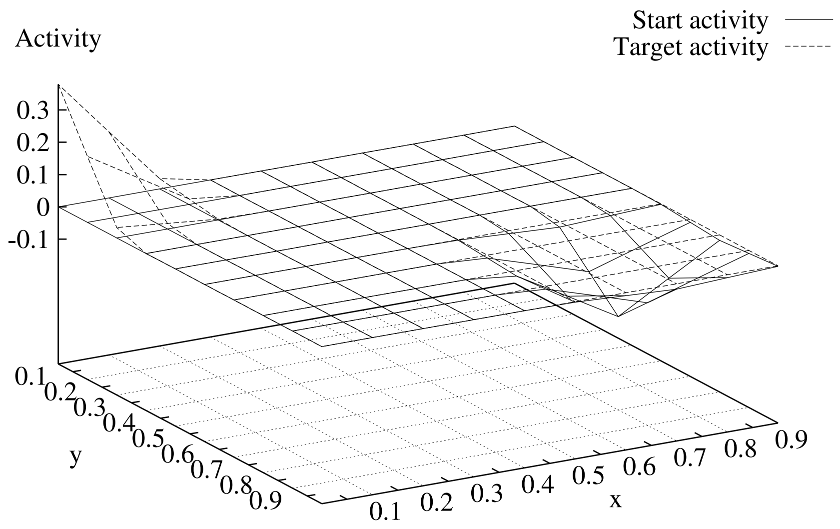

Figure 2: Initial activities

As can be seen the start activities are negative whereas the target activities are positive. The target and the plant are placed in the opposite corners, the neuronal discretizing layer being a rectangular array for ease of visualization.

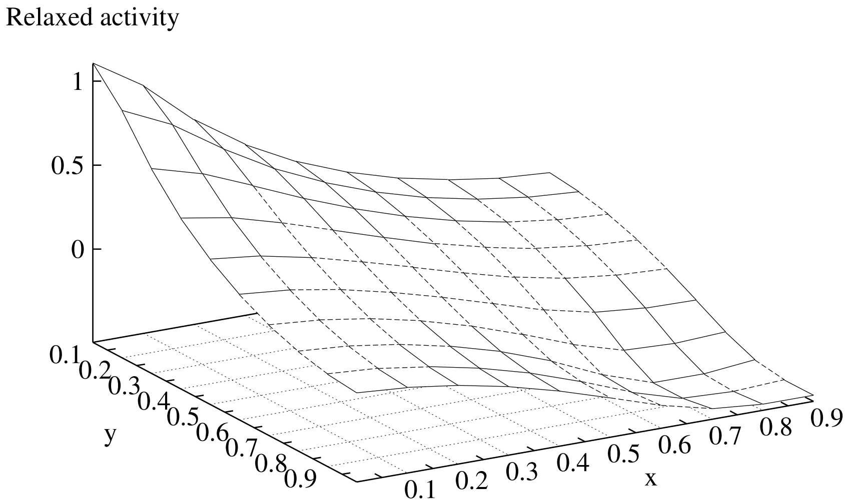

Figure 3: A typical relaxed neural activity field
The relaxed equilibrium activities, corresponding to the external flow shown in Fig. 2, have been plotted here. The plant should move along the gradient of this equilibrium field.

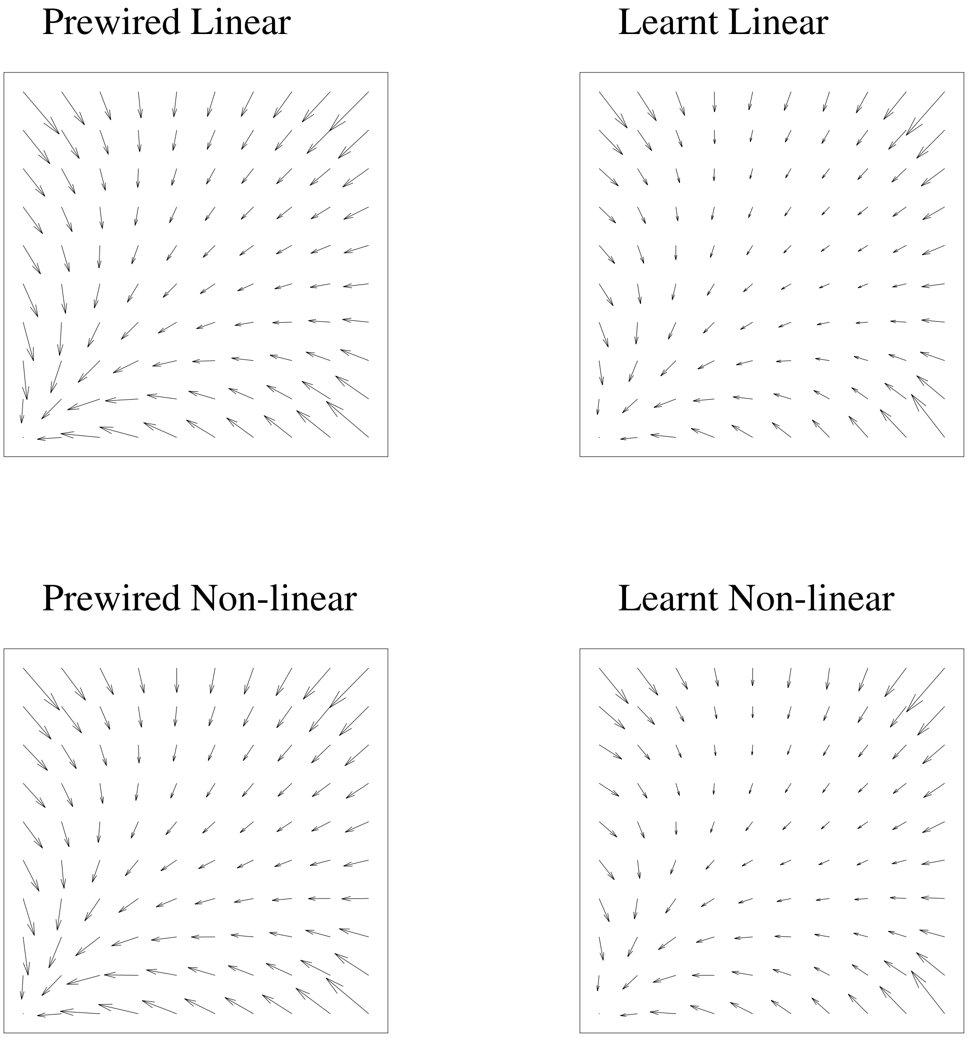

Figure 4: Speed vector fields
Upper row: linear plant, lower row: non-linear plant. Left column: prewired control command connections, right column: learnt control command connections. The arrows represent the plant-speed at different starting points when the target is placed in the lower left corner. The path of the plant started from any point is always tangential to the speed-vector field shown. This means that the plant moves to the target (lower left corner) in a more or less straight line. The magnitude of the arrows represent the magnitude of the plant speed-vector at different positions.

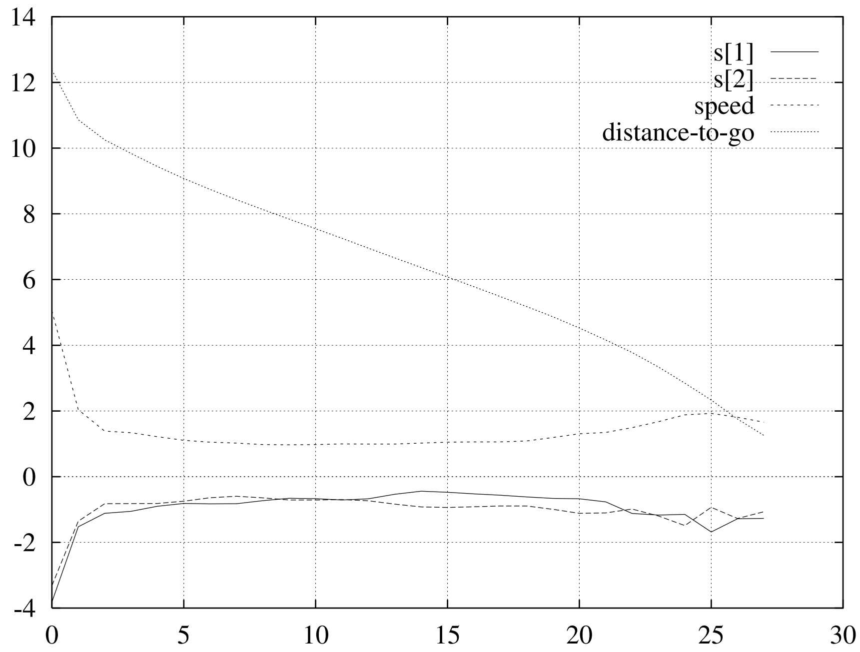

Figure 5: Distance-to-go and components of the speed-vector vs. time
The figure shows the distance left to reach the target, the plant speed and the two components of the speed-vector of the plant as a function of time while the plant is moving from the upper right to the lower left corner of the unit square. Path-length values have been multiplied by 10 for normalization.

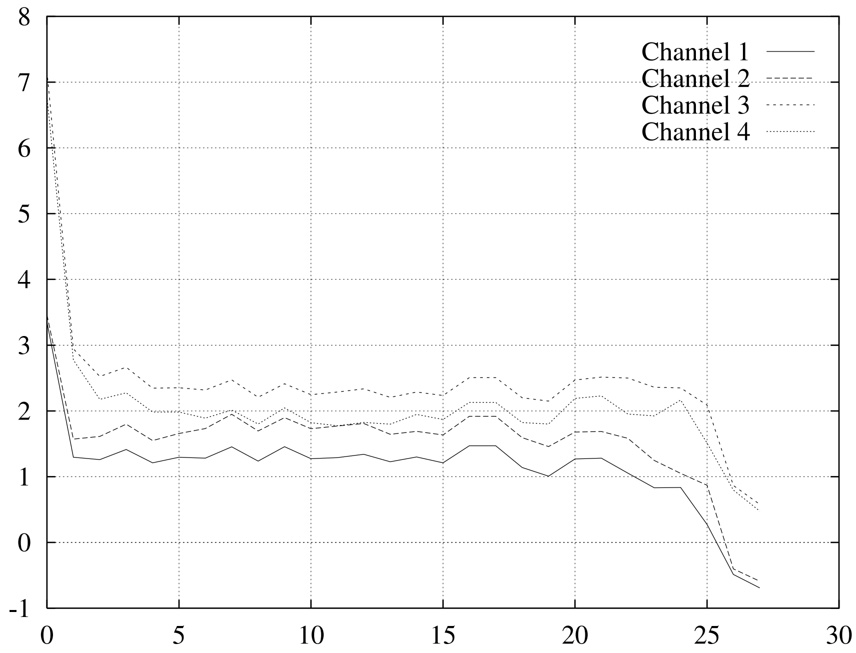

Figure 6: Components of control signals vs. time
The figure shows the output of individual control neurons as a function of time while the plant is moving from the upper right to the lower left corner of the unit square.

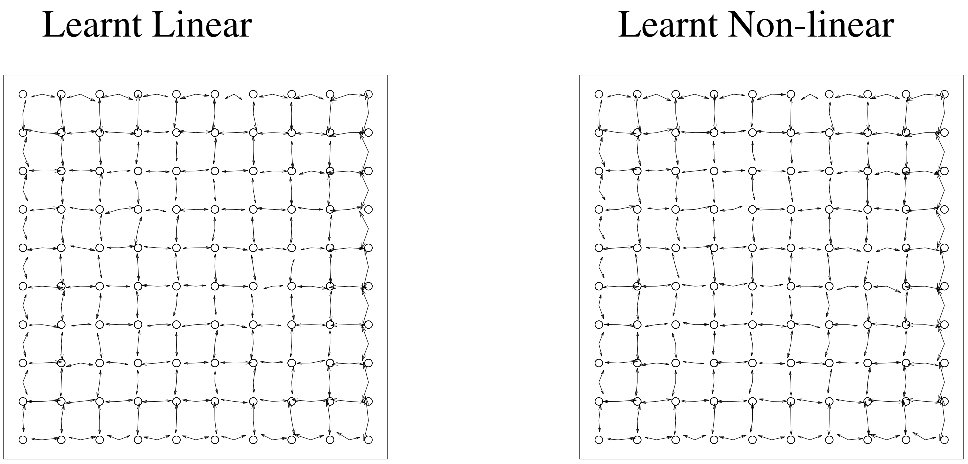

Figure 7: Learnt control command vectors for linear and non-linear plants.
The arrows start from interneurons situated between discretizing neurons. These represent the speed-vector of the plant provided the command vector of the corresponding interneuron is the control signal. As can be seen the differences between the two subfigures are quite minor. The strange similarities between the two parts are actually due to the same set of learning examples being adopted; with another choice the similarities disappear.

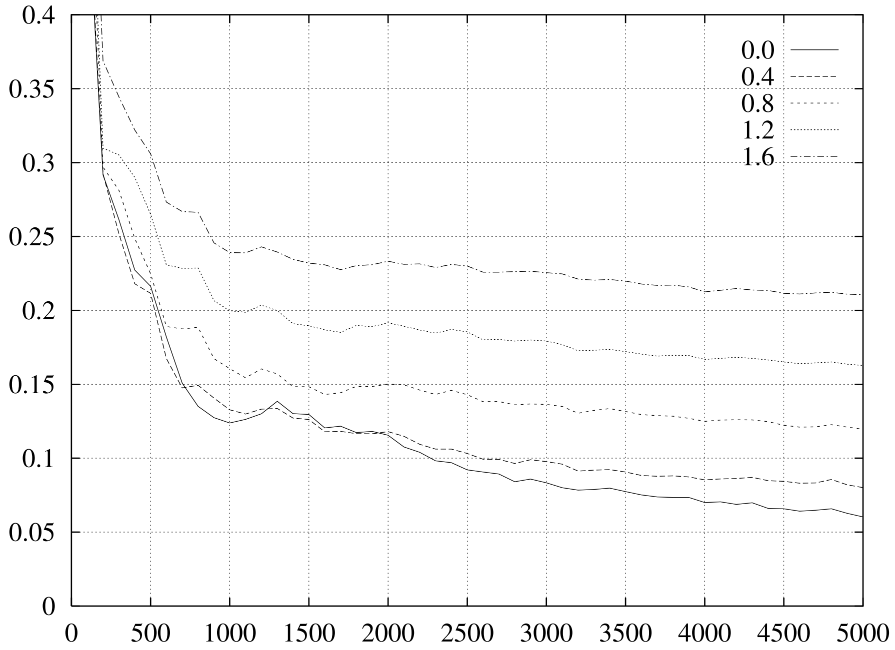

Figure 8: Learning curves vs. time

The curves depict the time-development of the error of the approximation to the inverse-dynamics for different types of non-linearities ($\alpha_{\max}$ values). The error is defined as the maximum deviation of the approximated inverse-dynamics from the corresponding theoretical value.

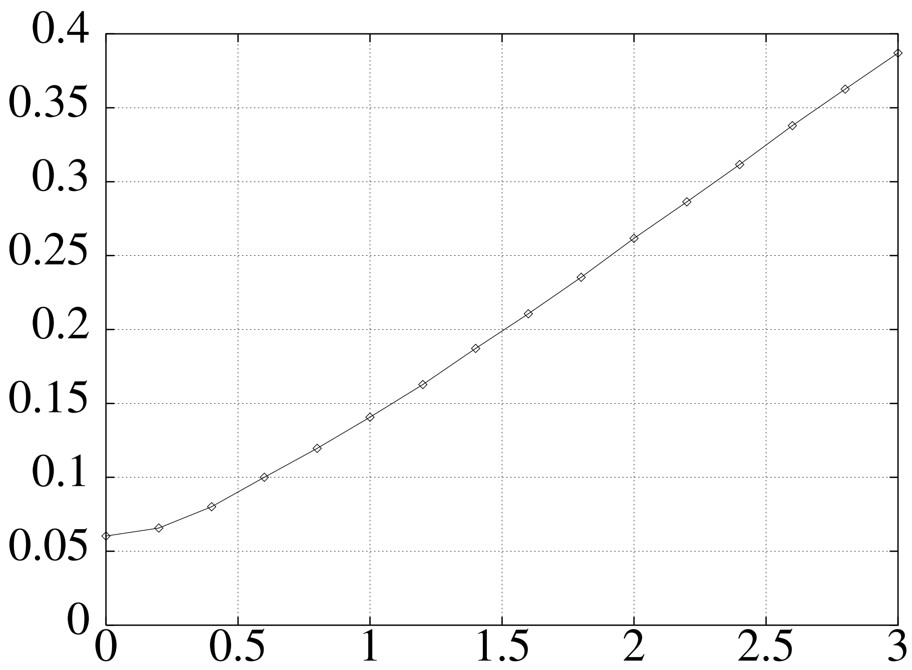

Figure 9: Error vs. non-linearity

Average error of the approximation to the inverse-dynamics as a function of the non-linearity present in the inverse-dynamics after completing identical training courses. After an initial phase the curve shows a linear dependence with the rate of growth of the non-linearity of the inverse-dynamics.