VISION-BASED ROBOT MOTION PLANNING USING A TOPOLOGY REPRESENTING NEURAL NETWORK

M. ZELLER, K. SCHULTEN
Beckman Institute for Advanced Science and Technology
University of Illinois at Urbana-Champaign
405 N. Mathews Avenue
Urbana, IL 61801, USA
email: zeller@ks.uiuc.edu - kschulte@ks.uiuc.edu

R. SHARMA
Department of Computer Science and Engineering
Pennsylvania State University
317 Pond Laboratory
University Park, PA 16802-6106, USA
email: rsharma@cse.psu.edu

The goal of integrating sensors into robot motion planning has incited recent research efforts. The *Perceptual Control Manifold* serves this goal extending the notion of the robot configuration space to include sensor space. In this paper, we develop a framework for sensor-based motion planning of robotic manipulators using the *Topology Representing Network* algorithm to develop a learned representation of the *Perceptual Control Manifold.* The topology preserving features of the neural network lend themselves to yield, after learning, a diffusion-based path planning strategy for flexible obstacle avoidance. We demonstrate the capabilities of topology preserving maps using an industrial robot simulator and a pneumatically driven robot arm (*SoftArm*).

1 Introduction

An important requirement for autonomous robotics is the ability to generate motion plans that achieve specified goals while satisfying environmental constraints. Motion planning is generally defined in terms of a configuration space or $\mathcal{C}$-space[9]. In most motion planning approaches, the $\mathcal{C}$-space is assumed to be known, implying a complete knowledge of both robot kinematics and obstacles. Uncertainty of these characteristics, however, is prevalent which makes such motion planning techniques inadequate for practical purposes. A sensing mechanism, for example, which uses video cameras and computer vision techniques, can help in overcoming uncertainties for guiding the motion of a robot[2]. However, to best utilize the sensor feedback given the limitations of these sensors, a robot motion plan should incorporate constraints from the sensor system as well as criteria for optimizing the sensor feedback.

Current robotic systems treat robot motion planning and control as two independent problems handled by different systems: (i) a motion planner determines a collision-free path to achieve a task, and (ii) the control system attempts to follow the planned path. The decoupling of sensor-based robot control from motion planning may not result in desirable plans. A feasible collision-free path may be hard to traverse, e.g., because of inadequate sensor feedback, whereas incorporating sensor constraints into motion planning may yield an easier path to control. To address this issue, a framework for motion planning was proposed in Refs.[23,20] that considers sensors as an integral part of the definition of the motion goal. The approach is based on the concept of a *Perceptual Control Manifold* (*PCM*), defined on the product of the robot $\mathcal{C}$-space and sensor space. The *PCM* provides a flexible way of developing motion plans that exploit sensors effectively. The configuration space framework can be extended to include sensor space constraints such as visibility, motion perceptibility and sensor singularity. The expected result of *PCM*-based motion planning is a path which is not only collision-free but also more efficient to control, i.e., optimizes the sensor feedback.

In many practical robotic systems the *PCM* cannot be derived analytically, since the exact mathematical relationship between configuration space, sensor space and control signals is not known. Even if the *PCM* is known analytically, motion planning may require the tedious and error-prone process of calibration of both the kinematic and imaging parameters of the system[27,6]. Instead of using the analytical expressions for deriving the *PCM* we propose, therefore, the use of a self-organizing neural network to learn the topology of this manifold. Once the *PCM* is learned, it can be used as a basis for sensor-based motion planning and control.

The rest of the paper is organized as follows. The general *PCM* concept is developed in Section 2; Section 3 describes the *Topology Representing Network* (TRN) algorithm[12] used to approximate the *PCM* and a diffusion-based path planning strategy which can be employed in conjunction with the TRN. The learned representation is then utilized for motion planning and control of a PUMA robot simulation (see Section 4) as well as on a pneumatic robot system (*SoftArm*), depicted in Section 5. In both cases, path control and flexible obstacle avoidance demonstrate the feasibility of this approach for motion planning in a realistic environment and illustrate the potential for further robotic applications.

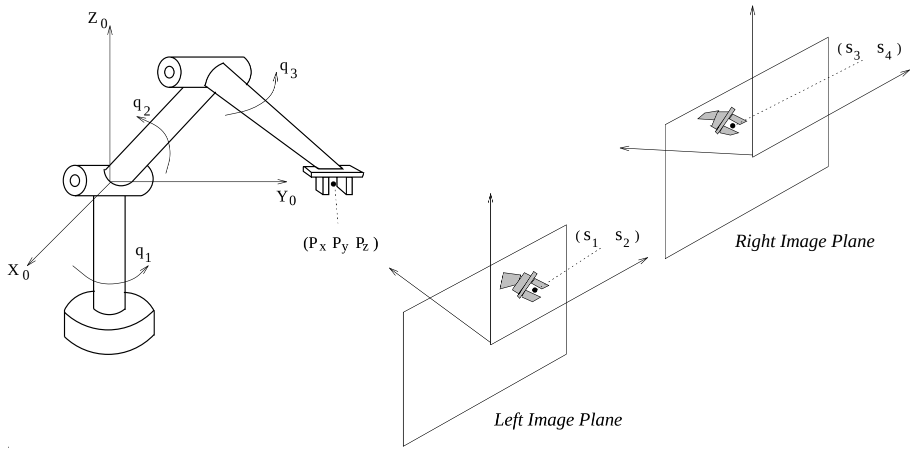

Figure 1: Schematic diagram of a 3-degree of freedom manipulator, and the mapping to the image feature space.

2 Perceptual Control Manifold

Sensors such as video cameras have a limited range of operation and work well only when the objects in view are suitably configured with respect to the camera [19,26]. Thus, to best utilize the sensor feedback, a robot motion plan should incorporate constraints from the sensor system as well as criteria for optimizing the quality of the sensor feedback. In Refs. [23,22], a method for incorporating the sensor constraints into motion planning was proposed with the help of a space called the *Perceptual Control Manifold* or *PCM* as discussed below.

The problem of motion planning of an articulated robot is usually defined in terms of the configuration space, $\mathcal{C}$ (or $\mathcal{C}$-space), which consists of a set of parameters corresponding to the joint variables of the robot manipulator. $\mathcal{C}$ is an n-dimensional manifold [9] for an n-degrees of freedom robot manipulator, i.e., $\mathcal{C} \equiv \mathcal{Q}_1 \times \mathcal{Q}_2 \times \ldots \mathcal{Q}_n \subseteq \mathbf{R}^n$, where $q_i \in \mathcal{Q}_i$ is a joint parameter (see Figure 1). The obstacles and other motion planning constraints are usually defined in terms of $\mathcal{C}$, followed by the application of an optimization criterium that yields a motion plan.

In vision-based control, the robot configuration is related to a set of measurements which provide a feedback about the Cartesian position of the end-effector using the images from one or more video cameras. We assume that this feedback is defined in terms of measurable image parameters that we call *image features*, s_i (see Figure 1). Before planning the vision-based motion, a set of m image features must be chosen. Discussion of the issues related to feature selection for visual servo-control applications can be found in Refs. [3,25,30]. The mapping from the set of positions and orientations of the robot tool to the corresponding image features can be computed using the projective geometry of the camera. Examples of commonly used projective geometry models include perspective, orthographic, or para-perspective projection models. Since the Cartesian position of the end-effector, in turn, can be considered to be a mapping from the configuration space of the robot, we can also define image features with a mapping from $\mathcal{C}$. Thus, an image feature can be defined as a function s_i which maps robot configurations to image feature values, $s_i : \mathcal{C} \rightarrow \mathcal{S}_i$. The set of all possible variations of the image features is termed *image feature space*, $\mathcal{S} \equiv \mathcal{S}_1 \times \mathcal{S}_2 \times \ldots \mathcal{S}_m$. A robot trajectory in configuration space will yield a trajectory in the image feature space.

Although we refer to vision, the discussion applies to any other sensor as well, and the term "image" is thus used for the generic sensor measurements. Examples of image features used in visual servo-control include image plane coordinates of a point, and other parameters of a line in the image, centroid, area, and other higher order moments of an image region. paper, we consider a hand/eye setup where the image features are derived from stationary cameras. Other examples where a *fixed-eye* configuration is used are given in Refs. [8,24]. For visual servoing examples in which an *eye-in-hand* setup is used, see Refs. [14,30], and for examples in which an *active* camera is used, see [19,21].

The *Perceptual Control Manifold* or *PCM* is defined as the manifold defined on the product space $\mathcal{C} \times \mathcal{S}$, or $\mathcal{CS}$-space. We know that an n-dimensional configuration space $\mathcal{C}$ maps to an m-dimensional feature space $\mathcal{S}$. Therefore, this mapping can be defined in terms of the vector-valued function $f : \mathcal{C} \rightarrow \mathcal{S}$ and results in an n-dimensional manifold embedded in an $(n+m)$-dimensional space.

For the robot in Figure 1, consider the variation of an image parameter, s_1, when a joint parameter, say q_1, is varied, while keeping the rest of the joints fixed. Without considering the joint limits for the time being, this would define an ellipse in the $\mathcal{Q}_1 \times \mathcal{S}_1$ space. Similarly, when two of the joints, say q_1 and q_2, are varied simultaneously, a hyper-ellipsoid will be defined in $\mathcal{Q}_1 \times \mathcal{Q}_2 \times \mathcal{S}_1 \times \mathcal{S}_2 \subseteq \mathbf{R}^4$. For ease of visualization, we project the corresponding *PCM* to $\mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{Q}_1 \subseteq \mathbf{R}^3$, as shown in Figure 2. Analogously, in higher dimensions, the *PCM* for a hand/eye setup is defined by varying all the joints and considering the parametric hypersurface defined in $\mathcal{Q} \times \mathcal{S}$ space.

A given robot configuration maps to exactly one point on the *PCM.* The corresponding image features are not necessarily unique for a given position,

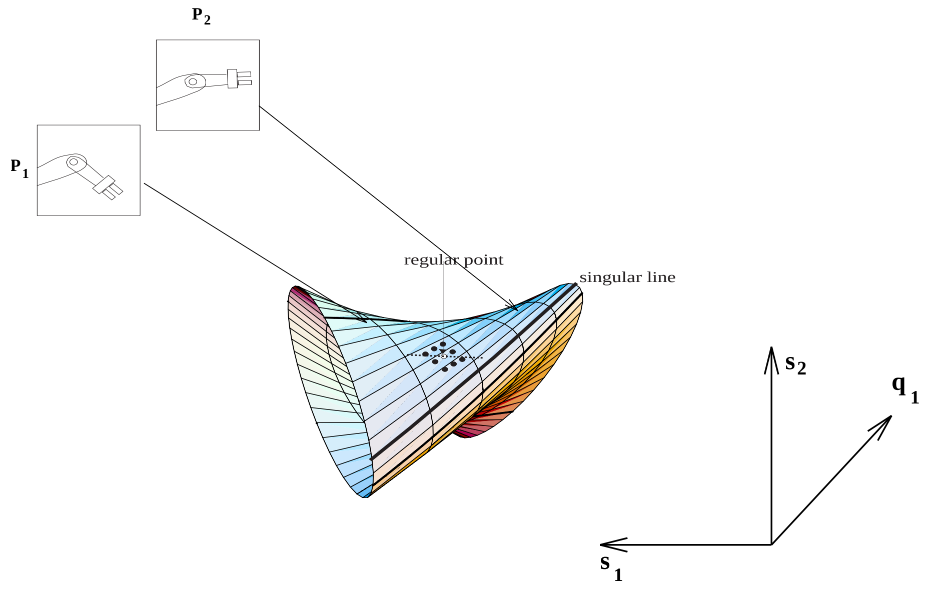

Figure 2: Positions of the manipulator mapped into the *PCM*. Each robot configuration P_i corresponds to exactly one point on the *PCM*.

but the additional representation of the joint establishes the uniqueness property needed for motion planning and control. Since the *PCM* represents both the control parameter and the sensor parameter, an appropriate control law can be defined on it [23]. Our concern in this paper is, however, on motion planning. An important characterization of the *PCM* for motion planning is the distinction between the *singular* and *regular* portions of the surface. The singularities of the *PCM* need to be identified so that they may be avoided during manipulation. The singularity constraints can be incorporated into the motion planning along with other constraints that are defined on the *PCM*.

A robot task can be defined as a problem of trajectory planning on the *PCM* from the initial position of the manipulator to some goal position. This motion planning requires the system to satisfy constraints presented by robot kinematics, the control system and the visual tracking mechanism. The use of the *PCM* makes the sensor constraints easier to express compared to a potentially awkward C-space representation. An example of such a constraint are image feature singularities [21].

With a complete knowledge of the robot kinematics and camera parameters, it would be possible to model the *PCM* analytically and carry out the motion planning on this space. However, as mentioned in Section 1, such an analytical model would be hard to derive under incomplete information, especially for a robot like the pneumatically controlled *SoftArm* that we use for our experiments in Section 5. For an industrial robot like the PUMA, described in Section 4, the kinematic model can be determined, but it is tedious to calibrate the camera setup especially if the cameras are frequently moved. This motivates us to aquire the *PCM* through a learning procdure and to subsequently use the learned space for sensor-based motion planning.

3 Topology Representing Networks for Motion Planning

For the path planning task described in the following sections, we seek to employ a neural network algorithm which is able to represent the topological characteristics of the *PCM*. Although several self-organizing processes have been suggested for forming topology-conserving maps, similar to the maps observed in, e.g., the visual, auditory and motor cortex [16], we will implement a recently introduced algorithm where the *a priori* knowledge of the input dimensionality is not crucial. Furthermore, the algorithm adjusts to the topological structure of a given input manifold $\mathbf{M}$ and forms, in contrast to most other self-organizing neural networks, always a perfectly topology preserving mapping. In many applications, the input manifold is a submanifold of a high-dimensional input space and may either be unknown or its topology may not be simple enough for prespecifying a correspondingly structured graph. For this purpose, the topology representing network [12](TRN) approach which we will outline in the following is best suited because it offers a flexible way to develop a discrete representation of the underlying data structure including neighborhood relationships. The TRN algorithm is a combination of the neural gas [11] vector quantization scheme and of a competitive Hebb-rule. While the neural gas part, a soft-competitive learning algorithm, distributes the input weights according to the input probability distribution, the competitive Hebb-rule preserves the topology by introducing neighborhood relations using a winner-take-all principle. Besides in visuo-motor control and path planning, this additional information is desirable in many applications [7]. For an extensive review on topology representing maps and biological brain function as well as an overview of different applications, see Ref. [18]. An extension of TRN, introducing a growing neural gas (GNG) network, where the number of neurons changes during the self-organizing process, is described in Ref. [4].

3.1 Topology Representing Network Algorithm

We start by initializing the input weights $\mathbf{w}_i$, for all units $i = 1 \dots N$, with random numbers and reset all connections to $c_{ij} = 0$. The data are assumed to be embedded in Euclidean space $\Re^D$ of dimension D, i.e. for weights holds $\mathbf{w}_i \in \Re^D$ and for input vectors $\mathbf{u} \in \Re^D$. For every input vector $\mathbf{u}$ the current ranking order

$$\|\mathbf{w}_0^{in} - \mathbf{u}\| \leq \|\mathbf{w}_1^{in} - \mathbf{u}\| \leq \dots \leq \|\mathbf{w}_{N-1}^{in} - \mathbf{u}\| \tag{1}$$

is determined, ordering the elements of the network by the Euclidean distance to the input signal. Using a soft-competitive learning rule, all input weights $\mathbf{w}_i^{in}$, $i = 1 \dots N$, are adjusted according to:

$$\mathbf{w}_i^{in}(t+1) = \mathbf{w}_i^{in}(t) + \gamma(r,t) \cdot (\mathbf{u} - \mathbf{w}_i^{in}(t)) \tag{2}$$

For the SoftArm in Section 5.1, we have to provide additional output weights

$$\mathbf{w}_i^{out}(t+1) = \mathbf{w}_i^{out}(t) + \gamma(r,t) \cdot (\mathbf{u} - \mathbf{w}_i^{out}(t)) \tag{3}$$

which will be used to link a desired control action to a specific sensory input. Adjustment of the weights $\mathbf{w}_i$ is guided by a monotonously decreasing function

$$\gamma(r,t) = \epsilon(t) \cdot e^{-r_i/\lambda(t)} \tag{4}$$

which depends on the current rank of a neuron r_i as determined in Equation 1 and the learning step t. In some cases it may be suitable to use a different $\gamma(r,t)$ for input and output weights. While

$$\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{max}} \tag{5}$$

determines the change in the synaptic weights, a neighborhood is represented by the function

$$\lambda(t) = \lambda_i(\lambda_f/\lambda_i)^{t/t_{max}} \tag{6}$$

Both $\epsilon(t)$ and $\lambda(t)$ are a function of time and gradually decrease as the learning step t approaches t_{max}. This guarantees an accelerated coarse adaption of all units at the beginning of the training procedure and provides the essential fine-tuning of the reference vectors $\mathbf{w}_i^{in}$ for $t \rightarrow t_{max}$. For the remaining parameters, we choose $\epsilon_i = 0.3$, $\epsilon_f = 0.05$, $\lambda_i = 0.2N$, $\lambda_f = 0.01$.

Up to this point the network has covered the neural gas vector quantization [1,13,11]. The next part of the algorithm introduces the competitive Hebb-rule which learns the topology by establishing neighborhood relations. For

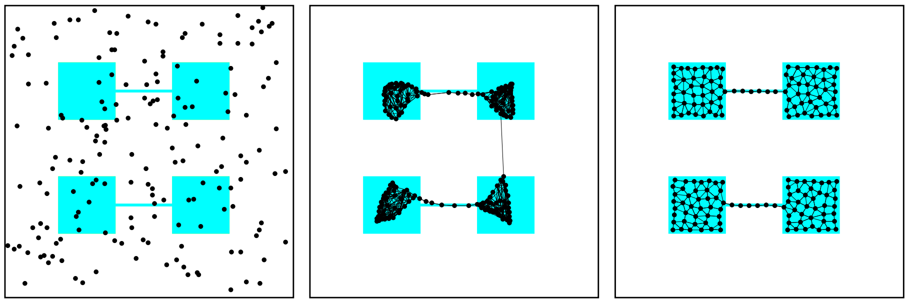

Figure 3: *Topology representing network:* When presented with samples from the grey shaded regions during the training cycle, the map develops from the initial distribution of neurons at $t = 0$ (*left*) through an intermediate distribution at $t = 10000$ (*middle*) to the final network at $t = t_{max} = 100000$ (*right*) which captures the topology of the input manifold.

this purpose, each connection of strength $c_{ij} \in [0, 1]$ is associated with an 'age' and the connection is removed if the age exceeds a certain lifetime without the corresponding connection $i \rightarrow j$ being activated[13]. Following a winner-take-all principle, for each learning step only the connection c_{01} between the units currently ranked 0 and 1 is altered. If $c_{01} = 0$ then we set $c_{01} = 1$ and the age of the connection is initialized with $t_{01} = 0$; if $c_{01} \neq 0$, we refresh the connection age, i.e., set $t_{01} = 0$. At the same time, we increase the age of all connections c_{0j} to $t_{0j} = t_{0j} + 1$ for all units j with $c_{0j} \neq 0$ and remove connections c_{0j} which exceed a given lifetime $t_{0j} > T(t)$, with $T(t) = T_i(T_f/T_i)^{t/t_{max}}$ and $T_i = 0.1N$, $T_f = 2N$.

To visualize the network dynamics, Figure 3 shows the development of a two-dimensional network. At the beginning, all neurons are initialized with random numbers, in this case $\mathbf{w}_i^{in} \in [0, \ldots, 1]$. During the learning process the network is presented with equally distributed random numbers drawn from the grey shaded areas and the vector quantization scheme gradually adapts the input probability distribution. The competitive Hebb rule introduces connections between the units which resemble the topology of the input manifold and, after a suitable training period, the algorithm detects and represents the two-dimensional squares of which two are connected by a one-dimensional line.

3.2 Diffusion-Based Path Planning

After the topology preserving map of the input manifold $\mathbf{M}$, which in our case is equivalent to the *PCM*, has been established, we want to generate a path from any initial position to a given target, e.g., to guide an end effector of a robot manipulator in the presence of obstacles within the workspace. We propose to use for this purpose the diffusion-based path finding algorithm suggested in Ref. [17] on the discrete network lattice in which the target neuron i_t is the source of a diffusing substance. The goal is to find a linked chain $i_{0,1,\ldots,n}$ on the graph leading from the current position $i_0 = i_c$ to the target position $i_n = i_t$.

To find the desired path, we define a function $f_i(t)$ on the nodes i of the network obeying the condition:

$$f_t(t) = 1 \qquad \forall t \tag{7}$$

and the relaxation dynamics:

$$f_i(t+1) = \frac{m}{N_i} \sum_{j \in F_i} f_j(t) \qquad \text{if} \quad i \neq i_t \tag{8}$$

The function $f_i(t)$, initially set to $f_i(0) = 0 \quad (i \neq i_t)$, represents the concentration at each node of the network and is held constant at the target node i_t while diffusing through the links of the network. F_i denotes the set of all nodes which are neighbors of i as defined by the network topology and N_i is the number of nodes in F_i. A flux can be defined as the concentration difference between two connected nodes and is directed towards the node with lower concentration. A value $m < 1$ corresponds to absorptive losses at the nodes. In order to avoid the trivial stationary solution we choose

$$m = \frac{N_i}{N_i + 1} \tag{9}$$

While other relaxation procedures can be used as well, this simple relaxation scheme serves our purpose and is computationally inexpensive, an important step towards real-time control, e.g., in the presence of moving obstacles.

The path leading from i_c to i_t can be found by starting at the current node i_c and choosing as the next step always the neighbor with maximal $f_i(t)$. Since the network graph is finite, the algorithm is guaranteed to terminate yielding a proper path $i_{c,1,2,\ldots,n-1,t}$. The path is short in the sense that it takes a route that maximizes the increase of $f_i(t)$ at each step.

To summarize, the motion plan can be generated as follows:

1. Read current position $\mathbf{u}_c$ and target position $\mathbf{u}_t$ in visual space.

2. Find best matching neurons $\mathbf{w}_c^{in}$ and $\mathbf{w}_t^{in}$.

3. Detect obstacles in vision space and eliminate connections to neurons i which are covered by an obstacle.

4. Define diffusion on network lattice and iterate until $f_i \neq 0$

$$f_i(t+1) = \begin{cases} 1 & \text{if } i = i_t \\ \frac{1}{N_i+1} \sum_{j \in F_i} f_j(t) & \text{else} \end{cases} \tag{10}$$

5. Follow steepest gradient of $f_i(t)$ from current unit i_c to target unit i_t.

In step 3 one can include other constraints, for example avoiding singularities in the sensor space [23]. It is not necessary to iterate step 4 until a stationary solution of the diffusion has been achieved. Instead, we can generate a path as soon as the concentration on the current node i_c is larger than zero. This might, however, render a re-computation of the diffusion necessary, if, for any reason, the current location is displaced into a position that has not been covered by the process yet. Finally, if the motion plan meets a given goal, movement can be initiated using the corresponding output values $\mathbf{w}_i^{out}$ of the map to generate the sequence of commands, e.g., to navigate a robot arm from start to target (see Section 5).

Other graph search algorithms and global optimization strategies can be applied to the learned representation of *PCM* as well. However, these will be computationally expensive, especially when complex obstacles are taken into account [9]. Our approach on the other hand is of complexity $O(N^2)$ with N being the number of nodes in the network and, in particular, it is independent of the number and shape of obstacles. Furthermore, the plan is resolution complete; only if the resolution of the discretized *PCM* manifold is not high enough to resolve a possible path, the algorithm will fail to find it. Following the steepest gradient of $f_i(t)$ from i_c to i_t eliminates the problem of local minima associated with many potential field path planning methods [9].

In Figure 4 we plot a sample path on a two-dimensional network ($N = 500$) from the upper left unit to the upper right which was obtained using the diffusion algorithm described above. Instead of learning a static topology including obstacles, we initially present the complete workspace during the training stage and dynamically map obstacles into the *PCM* after the representation of the workspace has been accomplished. This approach is more suited for a robotic manipulator operating in a changing environment, e.g., with obstacles placed at different locations within the workspace.

As a means of demonstrating the practical capabilities within an engineering framework for motion planning and control, the following sections will describe a PUMA robot simulator and the *SoftArm* robotic system as well as the implementation of the topology representing network algorithm for path planning on these systems.

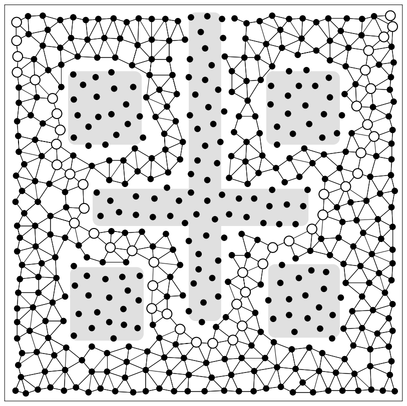

Figure 4: Two-dimensional *Topology Representing Network* after the learning has been finished. Obstacles have been identified and a sample path (open circles) from upper left unit to upper right unit of the map has been generated by the diffusion process.

4 Motion Planning for the PUMA Robot Manipulator

For off-line simulation and training, we developed a general purpose simulator for robot manipulator kinematics and visualization. Our simulator is intended for flexible testing and implementation of neural network control methods, e.g., to test the sensor-based motion planning approach described in Section 3. Furthermore, the simulator allows one to define arbitrary camera positions and to visualize the robot motion in 3d.

4.1 The PUMA Simulator

The design of the robot simulator is based on parallel distributed processing and socket-based interprocess communication which allows the different modules of the simulator to run in parallel on a network of workstations. In particular, it enables the neural network controller and the graphical front end to be executed on separate computers while communicating over a local area network. Our typical simulation setup consists of a TRN program running on a high performance workstation and the robot simulator, including visualization and vision processing, running on a workstation, best suited for fast graphics.

Figure 5 outlines the simulator, together with all its modules and communication paths. The robot controller, in our case a TRN with the sub-modules for vector quantization, competitive Hebbian learning and diffusion-based path planning, supplies the path plan and the corresponding control signals to ex-

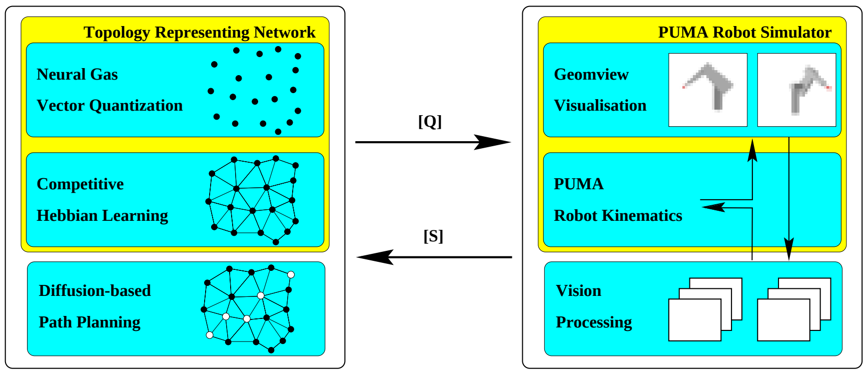

Figure 5: *PUMA robot simulator:* The TRN on the left side represents the robot controller and communicates via UNIX sockets with the PUMA kinematics module which handles the data transfer to Geomview, for visualization of the robot arm, and to the vision processing.

ecute the plan. The actual robot simulator, on the right side of Figure 5, includes the kinematic model of the PUMA, the Geomview package as graphical front-end and the vision processing subroutines.

The kinematics we employ here is based on the PUMA 562 industrial robotic manipulator, a 6-degree of freedom robot arm with a two-fingered gripper. All arm link coordinate parameters and the Denavit-Hartenberg representation have been published in Ref. [10]. The forward kinematics for the manipulator is calculated by a PEARL program which also handles the communication to and from the neural network controller as well as the data transfer between Geomview and the vision processing.

Visualization of the robot arm is based on Geomview [15], a public domain 3D visualization software [a]. This package can be used as a stand alone viewer for static objects or as a display engine for other programs, like our robot kinematics module which produce a dynamically changing geometry. Geomview allows the definition of multiple camera views and renders a fast 3D graphical representation of the robot arm. Although it is possible to query Geomview for the position of the robot geometry in ‘world coordinates’, we chose to use the Geomview camera frames as basis for vision processing. This emulates the situation we encounter for an experimental robotic setup where we would like to avoid any calibration tasks for the kinematic or vision parameters and solely rely on sensor feedback to generate the motion plan. Furthermore, the modular design of the simulator allows the kinematics subsystem to be replaced by a

[a] Geomview home page: `http://www.geom.umn.edu/software/geomview/`

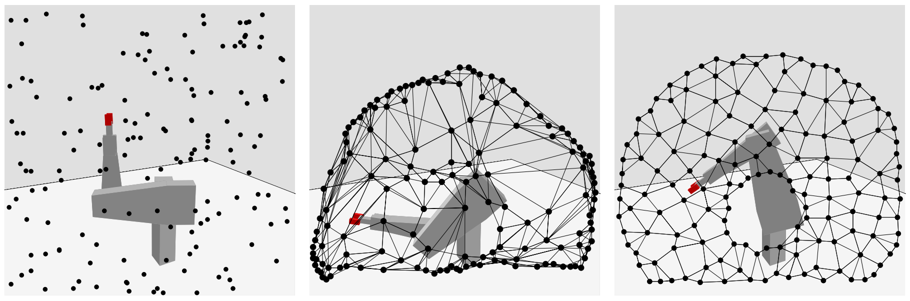

Figure 6: *Development of the topology representing network during the training:* (*left*) The initial distribution of neurons as initialized at $t = 0$. (*middle*) Intermediate form of the network after $t = 5000$ training moves. (*right*) The final network at $t = t_{max} = 50000$; the latter network resembles well the input manifold.

real robot. The vision processing programs will then receive data from actual color cameras. In our simulation, vision processing is based on color separation and thresholding in order to minimize the complexity, since our major concern here is adaptive control of the robot. Different colors are assigned, therefore, to the gripper and to obstacles, e.g., the gripper is drawn in red and the obstacles are blue while the rest of the scene is mostly colored in grey shades. Color separation yields the pixels covered by gripper and obstacles. For obstacle representation, we use all (x, y) pixel positions. The gripper position is idealized as one (x, y) value, calculated by the center of mass from the extracted blob.

4.2 Implementation

In previous research on neural network based visuo-motor control using a PUMA 562 robotic manipulator [29], we employed neighborhood preservation to average over the output of several adjacent units in order to speed up learning and to achieve a more accurate positioning. The present study seeks to exploit the topology to generate a motion plan from a current position to a given target satisfying several constraints. These constraints can include obstacles defined in $\mathcal{C}$-space, obstacles given through vision space and limitations of the camera feedback [21].

The *PCM*, as introduced in Section 2, is defined as the product of $\mathcal{C}$-space and sensor space $\mathcal{S}$. Therefore, two different types of information converge upon neurons within the network. Visual input $\mathbf{s} = (s_1 \dots s_n)^T$ is derived from the Geomview cameras; vision processing resolves the gripper location in each camera frame. Joint position of the manipulator, denoted by $\mathbf{q} = (q_1 \dots q_n)^T$, is generated by the network during the learning cycle and concatenated with $\mathbf{s}$ to the input vector $\mathbf{u} = (\mathbf{q}, \mathbf{s})^T$ for the TRN. Following a suitable training period, the topology of the neural network resembles the *PCM*. The current experiments focus on obstacle collisions of the robot's gripper only. Future studies will extend the control to avoid obstacles with the complete arm.

To better visualize how the neural network learns the topology of the *PCM* during the training cycle, we restrict our first example to one camera and two joints of the robot arm. Figure 6 shows the network projected onto the camera plane in three stages. First, we plot the initial distribution of the weight vectors at $t = 0$ when no connections exist, yet. After $t = 5000$ input vectors, generated by random movement of the manipulator in a two-dimensional plane, the network has not yet captured well the input manifold. Several connections are misplaced and the distribution of the weights is not optimal. Finally, after a set of $t = t_{max} = 50000$ training cycles, the TRN covers the complete input space and matches the *PCM* well. Given the learned representation of the *PCM*, we can use the network to generate a path plan from an initial gripper location to a given target point while avoiding obstacles which can be placed anywhere in the workspace of the robot arm. For static obstacles, we have to run the diffusion process only once and then follow the path. In case of moving obstacles, it is necessary to calculate an updated path at every step.

Figure 7 shows a 3D path, generated by the diffusion process. In this case, a network of $N = 1500$ neurons has been trained with a set of 80000 sample moves using three joints of the robot arm and the feedback from two cameras. Therefore, the input vector for the TRN consists of

$$\mathbf{u} = (\mathbf{q}, \mathbf{s}) = (q_1, q_2, q_3, s_1, s_2, s_3, s_4)^T \tag{11}$$

One obstacle is placed in the work space and mapped onto the learned representation of the *PCM* in order to eliminate connections which would lead to a collision of the gripper with the obstacle. The diffusion algorithm then generates the path and executes it by moving the arm along the planned trajectory. The vision feedback can also be used to correct deviations from the desired path. The total path length is 13 steps, of which six sample frames for both camera views are plotted in Figure 7. Interpolation between the nodes of the network can be introduced by a method described in Ref. [29] to further enhance the path resolution.

5 Motion Planning for the *SoftArm* Robotic System

In a next step, we port our path planning algorithm to an experimental robotic system which is quite different from the common industrial robot arm, e.g., the

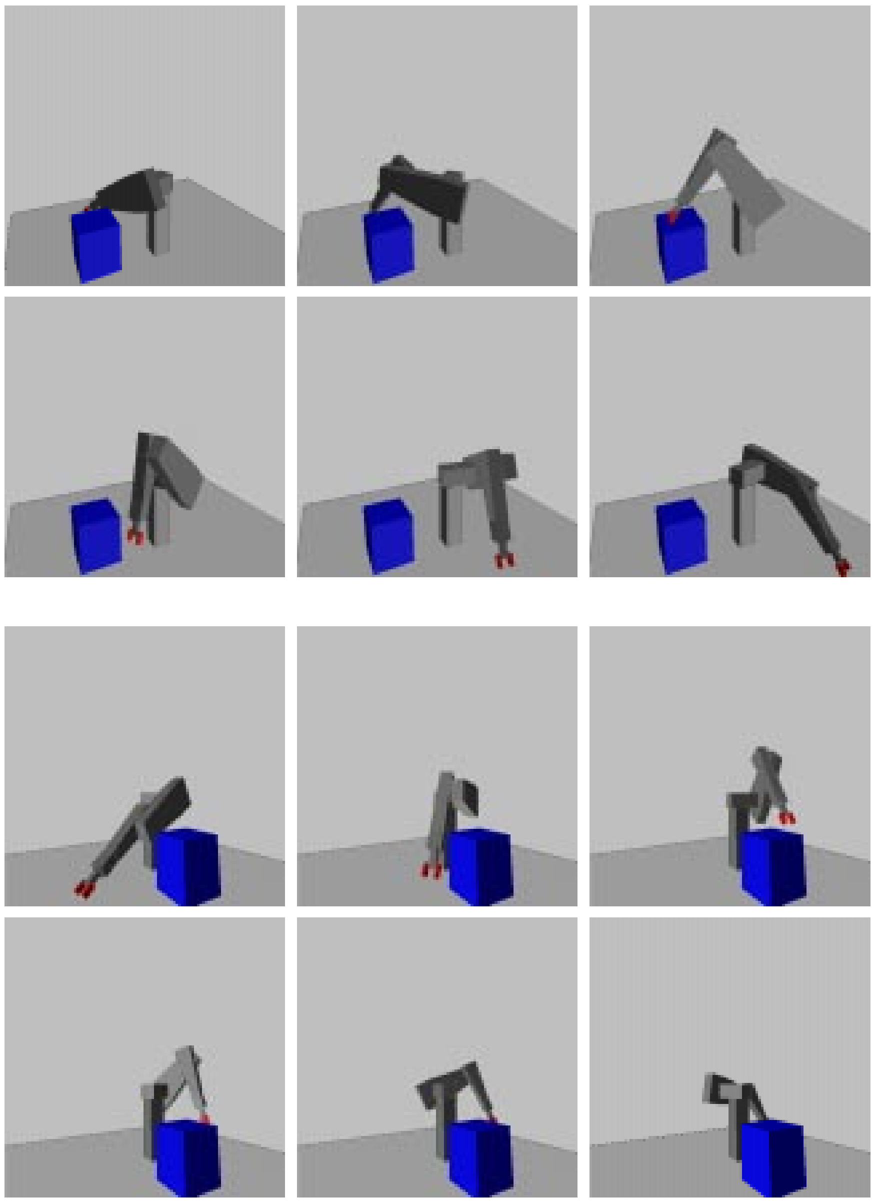

Figure 7: *3D path*, generated by the diffusion process algorithm, as seen by camera 1 (top 6 frames) and camera 2 (bottom 6 frames).

PUMA manipulator used in the previous section. The *SoftArm*, a pneumatically driven robotic manipulator manufactured by Bridgestone, is modeled after the human arm. It exhibits the essential mechanical characteristics of skeletal muscle systems employing agonist-antagonist pairs of *rubbertuators* which are mounted on opposite sides of rotating joints. Pressure differences drive the joints, average pressure controls the force (compliance) with which the motion is executed. This latter feature allows operation at low average pressures and, thereby, allows one to carry out a compliant motion of the arm. This makes such robots suitable for operation in a fragile environment, in particular, allows direct contact with human operators. The price to be paid for this design is that the response of the arm to pressure signals $(\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_N)^T$ and $(\Delta p_1, \Delta p_2, \ldots, \Delta p_N)^T$ cannot be described by *a priori* mathematical equations, but rater must be acquired heuristically. Furthermore, one expects that the response characteristics change during the life time of the arm through wear, after replacement of parts and, in particular, through hysteretic effects. In consequence, accurate positioning of the *SoftArm* presents a challenging problem and can only be achieved by an adaptive control mechanism. For a more detailed introduction to the mechanics of the *SoftArm* see Refs. [5,28].

5.1 The SoftArm *Robotic System*

The complete robot system, which is depicted in Figure 8, consists of the *SoftArm*, air supply, control electronics (servo drive units) and Hewlett Packard HP755/99 workstation which includes a serial interface, connected to the robot's servo drive units, and a video input card (Parallax Power Video 700Plus). The servo drive units provide the internal control circuitry of the robot, operate the servo valve units and send joint angle data, available from optical encoders mounted on each joint, to the computer. Visual feedback is provided by two color video cameras. For maximum flexibility, vision processing is implemented in software rather than in hardware. The use of a frame grabber to import the video signals in a JPEG encoded format minimizes the amount of data to be transferred between the video board and workstation memory. The location of the gripper is extracted from the video frames through a simple color separation, yielding one color component. This is then thresholded and the center of mass of the remaining image calculated. Coding the gripper in a certain color, e.g., red, allows us to weaken the workspace scenery restrictions in terms of background and lighting conditions while, at the same time, keeping the visual preprocessing simple and efficient.

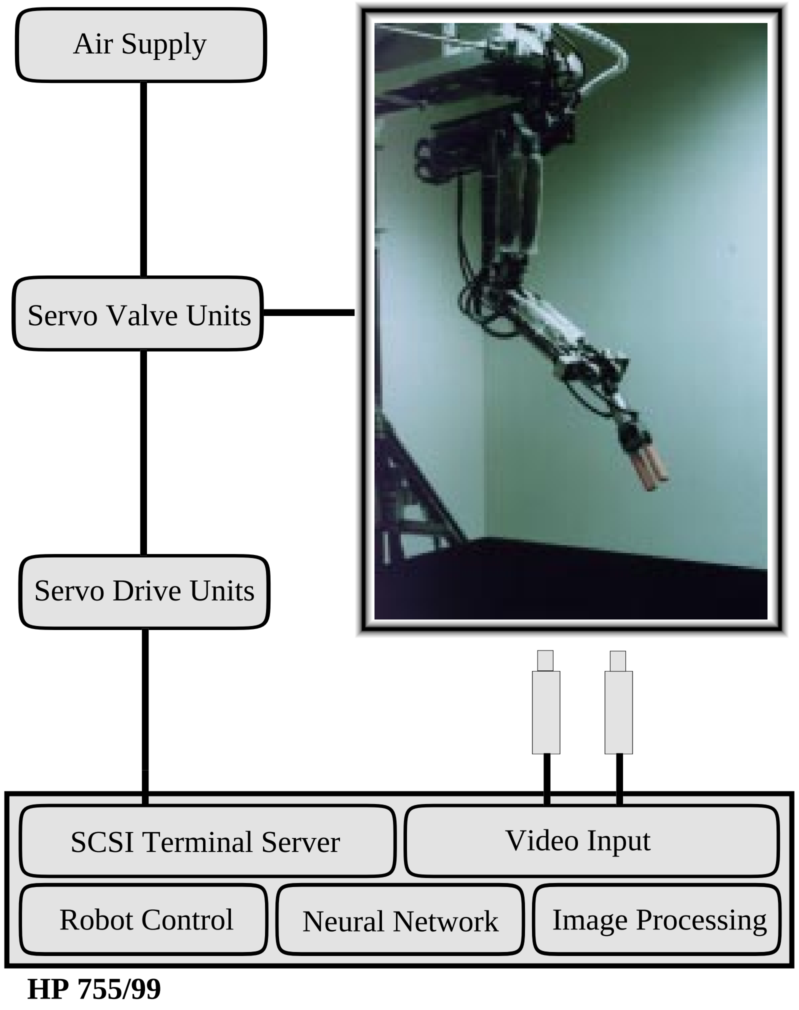

Figure 8: Diagram of the robot system, showing *SoftArm*, air supply, control electronics and workstation. The host computer includes a software layer (robot control, neural network and image processing programs) and the hardware components (serial interface and video input).

5.2 Implementation

We test our approach first in a two-dimensional environment, generating a motion plan in one camera plane. For this purpose, we use only two joints of the *SoftArm* to control the position. Visual input $\mathbf{s} = (s_1 \dots s_n)^T$ is derived from a color video camera; vision preprocessing resolves the gripper location in the video frames. Joint position of the manipulator, denoted by $\mathbf{q} = (q_1 \dots q_n)^T$, is derived from the feedback of optical encoders mounted on each joint. Following a suitable training period, the topology of the neural network resembles the *PCM*. In addition, the network provides the nonlinear mapping between the position in input space $\mathbf{u} = (\mathbf{s}, \mathbf{q})^T$ and the corresponding pressure commands $\mathbf{p}$ to achieve this configuration, in this case, between the 4D input vector $\mathbf{u} = (s_1, s_2, q_1, q_2)^T$ and the two-dimensional pressure vector $\mathbf{p} = (p_1, p_2)^T$.

A sample network is depicted in Figure 9 by plotting the visual components $\mathbf{s}$ of the 4-dimensional input vectors $\mathbf{w}_i^{in}$. This network was trained with a data set of 1000 random moves within a subset of the workspace and consists of $N = 75$ neural units. The left side shows the actual position in the robot's workspace as seen by the camera. On the right side, we use the learned representation to generate a motion plan from a start point to a given target. Both, start and target, are only given in visual space $\mathbf{s}$ (as is the obstacle), the corresponding encoder readings need not to be known. By selecting the best matching neurons for current position and target position in vision space the resulting neurons also provide the values for the encoder readings. This is possible, because $\mathbf{s}$ and $\mathbf{q}$ represent redundant information. The motion plan, shown in Figure 9, is generated by the diffusion algorithm exclusively in $\mathcal{CS}$-space to ensure a smooth motion in terms of joint angles.

Extending the algorithm to a three-dimensional workspace increases the information that needs to be processed by the network. The image feature space $\mathcal{S}$ is now represented by the position $\mathbf{s} = (s_1, s_2, s_3, s_4)^T$ of the gripper in two camera planes, the configuration space $\mathcal{C}$ is given by the encoder readings $\mathbf{q}$ of three joints, resulting in a 7D feature vector $\mathbf{u} = (s_1, s_2, s_3, s_4, q_1, q_2, q_3)^T$ and a three-dimensional output vector $\mathbf{p} = (p_1, p_2, p_3)^T$ respectively:

In this case, a network consisting of $N = 750$ neurons is trained with 5000 random moves within the workspace by sending random pressure values to the robot and observing the end effector position as well as reading out the encoder values. The sequence in Figure 10 shows a sample path. The robot arm is moving from the right side to the left side of the workspace while avoiding a collision with the obstacle placed in the middle. The training of the neural network was restricted to the lower portion of the workspace, so that no valid path over the obstacle exists in this case. Again, start and target location

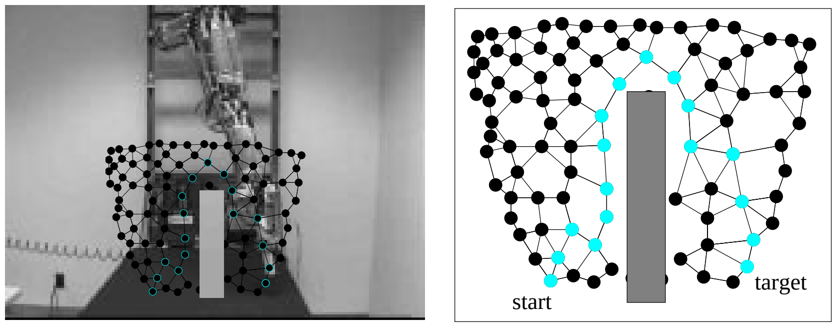

Figure 9: (*left*) *SoftArm* robot system, obstacle and network structure in the workspace as seen by the camera. The learning has been accomplished and the network represents the topology of the *PCM*. (*right*) Motion plan (grey units) generated by the diffusion process algorithm in $\mathcal{CS}$-space after start and target have been defined in vision space.

as well as obstacles are only known in vision space, while the motion plan is generated on the learned representation of the *PCM* by the diffusion procedure introduced in Section 3. In the 3D environment, a problem arises due to the discrete representation of the *PCM*. With our network of 750 neurons, the resulting path generated on the learned representation of the *PCM* is only 12 steps long which, in a more realistic environment, can only be seen as a rough motion plan. Nevertheless, it can be taken as a piecewise linear approximation of the final path and a basis for further path smoothing techniques[9]. Neural network interpolation strategies can be used to improve the accuracy as well[5].

6 Discussion

The presented simulations of a PUMA industrial manipulator and the implementation on the pneumatically driven *SoftArm* prove that learning the *PCM* with a topology representing neural network introduces a general framework for robot path planning. Sensing, e.g., in the form of video feedback, is automatically factored into the planning process, leading to a flexible way of visually controlling a robot manipulator. The motion plans, thus developed, can exploit properties of the sensed data and can also be linked to appropriately designed vision-based control laws. As our experiments in Sections 4.2 and 5.2 demonstrate, this method can be used to control robotic systems with multiple degrees of freedom in a 3D environment based on sensor data. The proposed path planning algorithm utilizes the topology preserving features of the neural network to dynamically map obstacles into the *PCM* and to establish a mo-

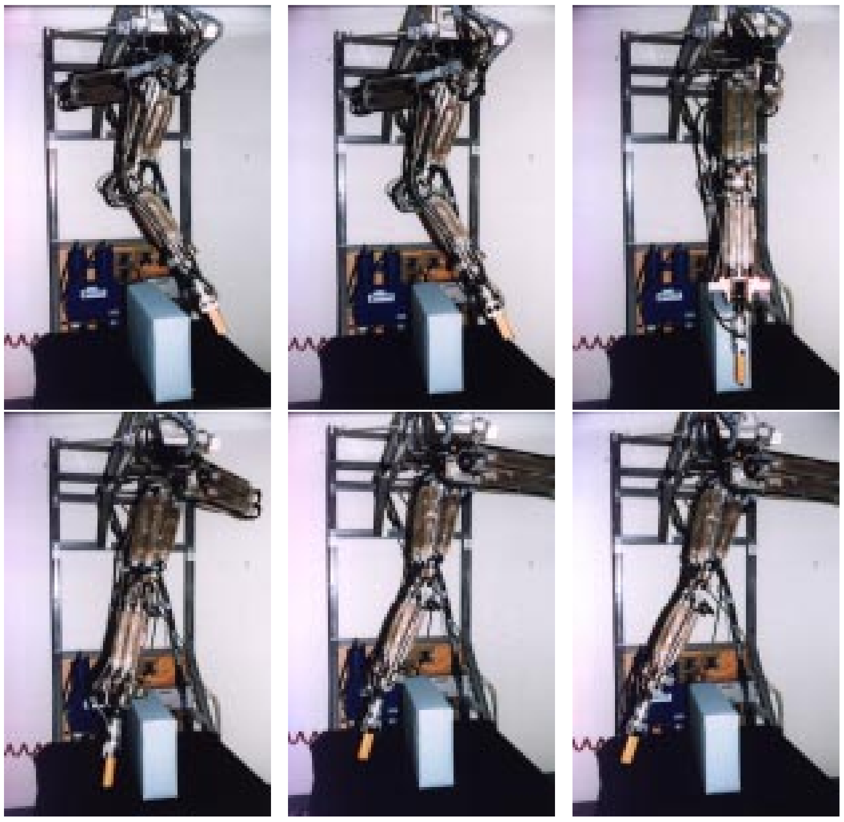

Figure 10: *3D path*, generated by the diffusion process. The successive frames show the robot arm moving from an initial configuration on the right side of the workspace to a given target position on the left side while avoiding a collision with the detected obstacle.

tion plan which prevents collisions of the gripper with detected obstacles. Not taking into account hardware specific limitations such as camera and encoder resolution, the accuracy of a motion plan is only limited by the network size. A larger number of neurons leads to a finer sampling of the underlying *PCM* topology and, hence, to a more detailed path, essential for an environment with complex obstacles. The parallel distributed PUMA simulator in Section 4.1 supports off-line training which enables us to use the simulator for extensive pre-training before implementing the network on a real PUMA robot. Since the TRN is able to handle even drastic changes of the robot geometry and adjust within a few hundred re-training cycles[29], it will also adjust to changes in the camera positions relieving one from tedious camera calibration task.

In contrast to the industrial PUMA robot system, the *SoftArm* has not been designed to facilitate accurate posture control. A simulator environment is not available and the mechanics of the arm confines the size of our data base to a few thousand moves, leading to a coarse representation of the manifold. Hence, the number of training samples in our respective experiments was much smaller than in the simulations for the industrial robot simulator. Future work will have to address the discretization effect in higher dimensions, as introduced by the use of redundant degrees of freedom, to achieve a finer path control. The discretizing effect, that results from the use of small numbers of neurons to map a high dimensional input space, can be alleviated by introducing interpolation strategies[5] which also improve fine motion control. Furthermore, future research should focus on extending the presented framework to include active sensing as well as collision free motion planning of highly redundant robotic manipulators within an uncertain and changing environment.

Acknowledgments

The authors wish to thank Herry Sutanto for providing the initial PUMA kinematics module used for the Geomview visualization. This work was supported by the Roy J. Carver Charitable Trust (M.Z. and K.S.) and by the U.S. Army Research Laboratory under Cooperative Agreement No. DAAL01-96-2-0003 (R.S.). Simulations were carried out at the Resource for Concurrent Biological Computing at the University of Illinois, funded by the National Institute of Health (Grant P41RR05969) and by the National Science Foundation (Grants BIR-9318159 and ASC-8902829)

References

1. Stanislav G. Berkovitch, Philippe Dalger, Ted Hesselroth, Thomas Martinetz, Benoît Noël, Jörg A. Walter, and Klaus Schulten. Vector quantization algorithm for time series prediction and visuo-motor control of robots. In W. Brauer and D. Hernandez, editors, *Verteilte Künstliche Intelligenz und kooperatives Arbeiten*, volume 291 of *Informatikfachberichte*, pages 443–447. Springer, Berlin, 1991.
2. P. I. Corke. Visual control of robot manipulators—a review. In K. Hashimoto, editor, *Visual Servoing*, pages 1–32. World Scientific, 1993.
3. J. T. Feddema, C. S. George Lee, and O. R. Mitchell. Weighted selection of image features for resolved rate visual feedback control. *IEEE Transactions on Robotics and Automation*, 7:31–47, 1991.
4. B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advanced in Neural Information Processing Systems 7*, pages 625–632, Cambridge, MA, 1995. MIT Press.
5. T. Hesselroth, K. Sarkar, P. van der Smagt, and K. Schulten. Neural network control of a pneumatic robot arm. *IEEE Transactions of System, Man and Cybernetics*, 24(1):28–37, 1994.
6. J. M. Hollerbach. A survey of kinematic calibration. In O. Khatib, J. J. Craig, and T. Lozano-Pérez, editors, *Robotics Review 1.* MIT Press, Cambridge, MA, 1989.
7. T. Kohonen. *Self-Organizing Maps.* Springer, New York, 1995.
8. A. J. Koivo and N. Houshangi. Real-time vision feedback for servoing robotic manipulator with self-tuning controller. *IEEE Transactions on Systems, Man, and Cybernetics*, 21:134–142, 1991.
9. J. C. Latombe. *Robot Motion Planning.* Kluwer Academic, Boston, MA, 1991.
10. C. S. G. Lee. Robot arm kinematics. In C. S. G. Lee, R. C. Gonzales, and K. S. Fu, editors, *Tutorial on Robotics, Volume 1*, pages 47–65. IEEE, 1983.
11. T. Martinetz and K. Schulten. A 'neural gas' network learns topologies. In *Proceedings of the International Conference on Artificial Neural Networks, Helsinki, 1991.* Elsevier Amsterdam, 1991.
12. T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.
13. Thomas M. Martinetz, Stanislav G. Berkovich, and Klaus Schulten. "Neural gas" for vector quantization and its application to time-series

prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569, 1993.

14. N. P. Papanikolopoulos, P. K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of vision and control. *IEEE Transactions on Robotics and Automation*, 9(1):14–35, 1993.
15. M. Philips. *Geomview: 3D Visualization Software.* 1300 South 2nd St, Suite 500, Minneapolis, MN 55454, U.S.A., 1996. [http://www.geom.umn.edu/software/geomview/].
16. H. Ritter, T. Martinetz, and K. Schulten. *Textbook: Neural Computation and Self-Organizing Maps: An Introduction.* Addison-Wesley, New York, revised English edition, 1992.
17. H. Ritter and K. Schulten. Planning a dynamic trajectory via path finding in discretized phase space. In *Parallel Processing: Logic, Organization, and Technology*, volume 253 of *Lecture Notes in Computer Science*, pages 29–39. Springer, 1987.
18. K. Schulten and M. Zeller. Topology representing maps and brain function. In *Nova Acta Leopoldina NF 72*, volume 294, pages 133–157. Deutsche Akademie der Naturforscher, 1996.
19. R. Sharma. Active vision for visual servoing: A review. In *IEEE Workshop on Visual Servoing: Achievements, Applications and Open Problems*, May 1994.
20. R. Sharma, J-Y. Hervé, and P. Cucka. Dynamic robot manipulation using visual tracking. In *Proc. of the 1992 IEEE International Conference on Robotics and Automation*, pages 1844–1849, May 1992.
21. R. Sharma and S. Hutchinson. Optimizing hand/eye configuration for visual-servo systems. In *Proc. IEEE International Conference on Robotics and Automation*, pages 172–177, May 1995.
22. R. Sharma and H. Sutanto. Unifying configuration space and sensor space for vision-based motion planning. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3572–3577, April 1996.
23. R. Sharma and H. Sutanto. A framework for robot motion planning with sensor constraints. *IEEE Transactions on Robotics and Automation*, 13(1), 1997.
24. S. B. Skaar, W. H. Brockman, and R. Hanson. Camera-space manipulation. *International Journal of Robotics Research*, 6(4):20–32, 1987.
25. H. Sutanto and R. Sharma. Global perfomance evaluation of image features for visual servo control. *Journal of Robotic Systems*, 13(4):243–258, April 1996.
26. K. A. Tarabanis, P. K. Allen, and R. Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on Robotics and Automation*,

11:86–104, 1995.

27. R. Y. Tsai. Synopsis of recent progress on camera calibration for 3d machine vision. In *Robotics Review 1.* MIT Press, Cambridge, MA, 1989.
28. P. van der Smagt, F. Grön, and K. Schulten. Analysis and control of a rubbertuator arm. *Biological Cybernetics*, 75:433–440, 1996.
29. J.A. Walter and K. Schulten. Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, 4(1):86–95, 1993.
30. L. E. Weiss, A. C. Sanderson, and C. P. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal of Robotics and Automation*, 3:404–417, 1987.