NARM: A NEURAL CONTROLLER FOR COLLISION-FREE MOVEMENT OF GENERAL ROBOT MANIPULATORS

Daryl H. Graf and Wilf R. Lalonde

SCS-TR-133

April, 1988

This research was supported by the Natural Sciences and Engineering Research Council of Canada.

School of Computer Science, Carleton University
Ottawa, Canada K1S 5B6

NARM: A Neural Controller For Collision-Free Movement of General Robot Manipulators

Daryl H. Graf and Wilf R. LaLonde

Carleton University, Ottawa, Canada, K1S 5B6

Abstract: This paper presents an approach for the collision-free control of general robot manipulators moving among a changing set of obstacles. The Neural Adaptive Robot Manipulator (NARM) controller, based on a layered, neural network architecture, adapts to the specific eye/hand and arm/body kinematics of any arbitrarily shaped robot during an initial, unsupervised training phase. After training, the robot selects a target point by "glancing" at it and the controller moves the end-effector into position. Collision-free movement is produced regardless of the number or arrangement of obstacles in the workspace. Moreover, no additional learning is required if the obstacle set is changed. This approach has several advantages over traditional algorithmic solutions and extends previous work on neural manipulator control. Results of a simulation are presented.

I. Introduction

Traditionally, designing a robot control system involves two steps: first, a set of kinematic equations which express the physical constraints of the robot are derived, and second, a computer program employing these equations is written to generate arm configuration sequences that move the robot's end-effector from its current position to a target position. While these programs work well in the laboratory, they often suffer from serious limitations when applied to realistic environments. This is because the real world is a hostile place for robots. Wear and tear on mechanical parts changes the kinematics of manipulators and sensor characteristics tend to wander with time. When such changes occur, the control program must be updated or the robot must be maintained. Realistic workspaces also often contain sets of unpredictable obstacles. Avoiding collisions with these obstacles adds significantly to the complexity of programs and can slow response time if the obstacles have complicated shapes or change position often. Nature on the other hand has designed systems which deal with these kinds of problems remarkably well. The key to this success is a high degree of autonomous adaptive learning.

Neural architectures offer an alternative approach to the design of robot control systems. Like biological systems, neural architectures support a high degree of adaptability and are capable of learning sensory-motor constraints from training examples. Various authors have investigated neural architectures for manipulator control (Bullock and Grossberg[3], Grossberg and Kuperstein[8], Kuperstein[12], Tsutsumi[18,19], Elsley[20]). We extend this work by developing an architecture that, in addition to learning kinematic constraints for eye-hand coordination, also learns obstacle avoidance constraints that can be used to rapidly find collision-free movements of the arm. In this paper we show how the architecture is able to learn to generate, in constant time, many alternative solution configurations which touch a target point, and we show how learned collision constraints may be used to find a collision-free arm movement. Collision avoidance constraints are learned once. No additional learning is required if the obstacle set changes. For a more complete treatment and error analysis, see Graf[6]. The accuracy of our approach, as with most neural systems, is dependent on the number of neural processors and interconnections available. Although the number of processors increases rapidly with the complexity and precision of the manipulator, we are encouraged by the work of Y. Abu-Mostafa and D. Psaltis (Abu-Mostafa and Psaltis[1] and Psaltis et al[16]) which demonstrates the feasibility of optical systems supporting thousands of processors and millions of connections. We also point out that, for many applications, small networks capable of less precision are accurate enough to plan broad movements of the manipulator. These networks can be combined with other mechanisms to refine this movement near a target.

2. System Architecture

Figure 1 is a schematic representation of the architecture of our adaptive robot. In order to simplify the presentation, we limit our discussion to a 2-dimensional robot with a 2 degree-of-freedom, revolute-jointed arm. It should be emphasized however that our approach generalizes to 3-dimensional robots and arms of any number of degrees-of-freedom. In addition, the arm links may be of arbitrary shape and the arm joints may be revolute and/or prismatic. The robot consists of a body B of arbitrary shape, a J degree-of-freedom manipulator, and a sensory pallet containing stereo cameras or range finders C_1 and C_2 mounted on the robot body at a point that provides a clear line of sight to the end effector E. C_1 and C_2, hereafter called the eyes of the robot, can be turned independently. Joints J_1 and J_2 each have joint angle sensors which output the amount, θ_1 and θ_2, that the associated links L_1 and L_2 are offset from their rest positions. We will refer to the vector θ as a **joint angle vector** or **arm configuration**. When the line of sight of each eye is centered on a workspace point p_g, called a **point of gaze**, the eye motor displacement angles, Ψ_1 and Ψ_2, uniquely define its direction and distance. A D-dimensional workspace will require a minimum of D displacement angles to uniquely specify each point of gaze. We will refer to Ψ as an **eye angle vector** or **eye configuration**.

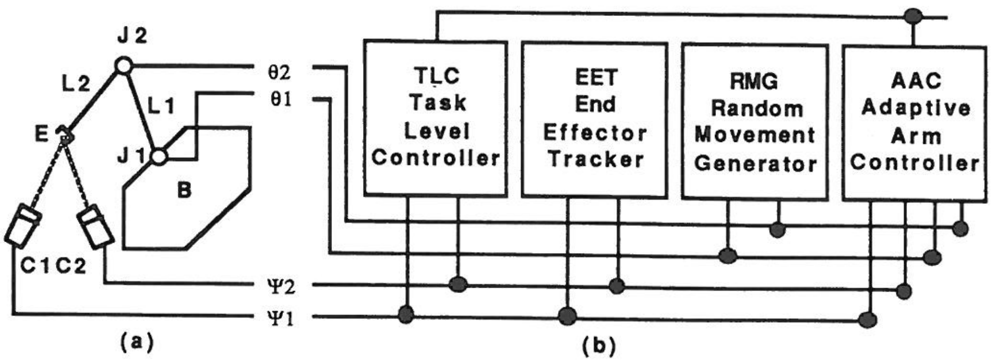


Figure 1: The Adaptive Robot Architecture

The four control subsystems of the robot are shown in Figure 1b. The **task level controller** (TLC) is a high-level task planner that selects a target point and orients the eyes toward it. Glancing at the point directs the arm to touch it with the end-effector. Targets are selected relative to the current sensory view rather than by referring to an objective, universal coordinate system. The **end–effector tracker** (EET) is a primitive controller that orients the eyes toward the tip of the end-effector whenever the TLC is not otherwise determining the point of gaze. The EET is ignorant of the kinematics of the arm. It merely tracks a moving light source or pattern that is attached to the end-effector. During learning, the **random movement generator** (RMG) increments each joint angle by small random amounts. We assume that, if the arm is brought into contact with an obstacle, force sensors prevent it from being damaged until its random movements free it. The EET will track the end-effector as the arm moves randomly through the workspace. The **adaptive arm controller** (AAC), the focus of this paper, receives joint angle feedback θ from the arm, eye angle feedback Ψ from the sensory pallet, and information regarding the positions of objects in the workspace. Ψ can also be taken from the TLC, rather than directly from the eye angle sensors, allowing the TLC to propose target points that the eyes are not currently viewing.

Initially, the AAC has no knowledge of the kinematics of the eye/arm/body combination it is associated with. During a preliminary training phase, it must learn to associate eye angle vectors with one or more **target arm configurations** that place the end-effector within an allowable distance from the target point as shown in Figures 2a,b. Once training is complete and the TLC has selected a target, the AAC must generate a sequence of **joint control vectors** that move the arm from its current configuration to one of the target configurations. A joint control vector consists of J joint angles that the joint motors are to assume. Learned obstacle avoidance constraints must prevent the AAC from generating undesired arm configurations such as configuration 1 in Figure 2c. In short, the controller must learn a "sense of space" (Kuperstein[12]) from its own sensory-motor experiences.

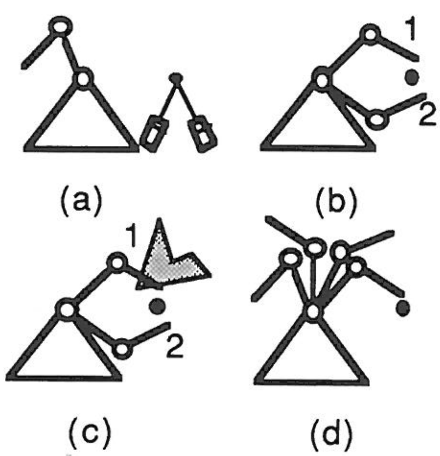


Figure 2: Learning Arm-Eye Configurations

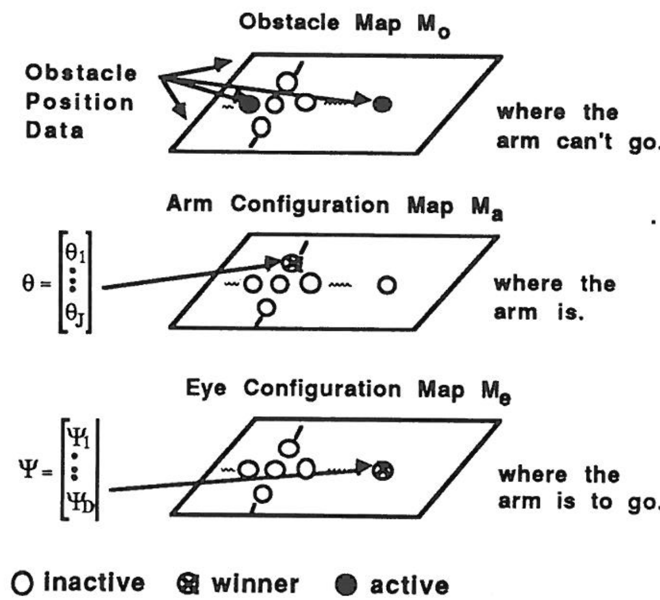


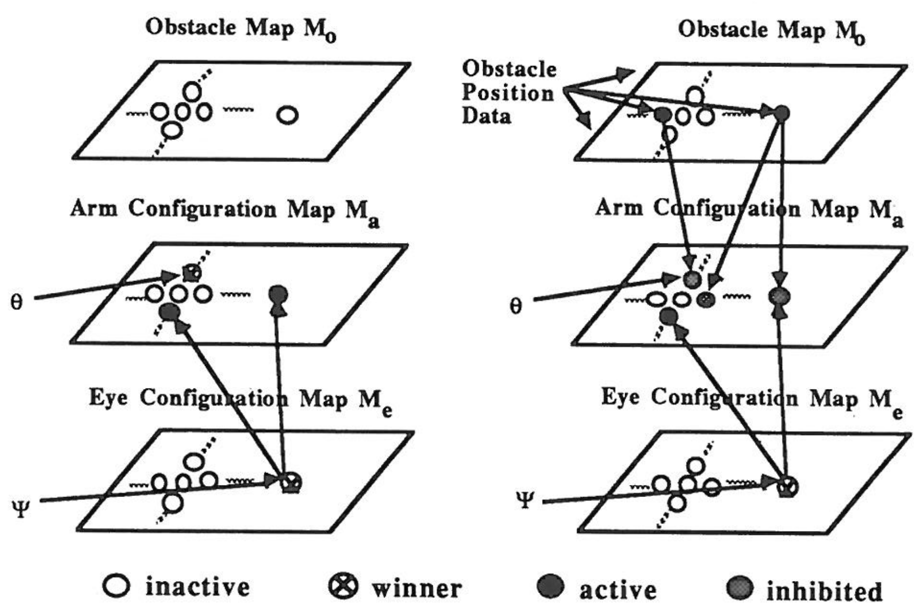
Figure 3: The Adaptive Arm Controller Architecture

3. The Adaptive Arm Controller Architecture

The AAC consists of three layers of neural processors as shown in Figure 3: an **obstacle map** M_o, an **arm configuration map** M_a, and an **eye configuration map** M_e. M_o represents regions of the workspace where the arm cannot go, M_a represents where the arm currently is and where it can be (the configuration of the arm determines the position of the end effector), and M_e represents the place where the arm is intended to go (the target point of gaze that the end effector must touch).

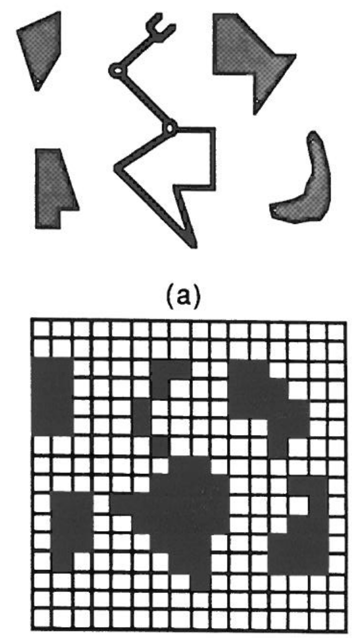
3.1 How The AAC Is Used To Move The Arm To A Target

Before considering a detailed view of each map, we offer an intuitive explanation of the steps that move the end-effector to the point being looked at. Each workspace point occupied by an obstacle activates a processor in M_o, the current arm configuration activates one processor in M_a, and the target point of gaze activates one processor in M_e. **Inter-map** connections associate regions of space in M_o with arm configurations in M_a, and eye configurations in M_e with arm configurations in M_a. Figures 4a,b illustrate these connections in two parts to make the presentation clear. In practice however, these two stages of processing occur in parallel. As shown in Figure 2b, each eye configuration corresponds to one or more target arm configurations. Consequently, as shown in Figure 4a, connections from the active processor in M_e cause one or more target configuration processors in M_a to be activated (black circles). Figure 4b shows that connections from active processors in M_o inhibit those processors in M_a (gray circles) that correspond to arm configurations intersecting obstacles (e.g., configuration 1 in Figure 2c). At this point, the output signals of processors in M_a represent the current arm configuration, the possible target arm configurations and those arm configurations which result in collisions with obstacles.



(a) Connections from M_e to M_a (b) Connections from M_o to M_a

Figure 4: Inter-Map Connections



(b)

Figure 5: The Obstacle Map

If two important conditions that we discuss in a moment hold, the final step of finding an arm movement sequence (not shown here), is simply a matter of finding a path — a chain of neighbouring processors — through M_a, that does not include an inhibited processor, extending from the current arm configuration processor to any un-inhibited target configuration processor. The sequence of arm configurations associated with processors on this path will safely lead the arm to the targetted point of gaze. Path finding is a well understood computational problem and many efficient serial and parallel algorithms already exist to find paths with desired characteristics. For example, if the arm configuration map is also a systolic array, we may use a modified version of the algorithm by Miller and Stout[15] to find a minimal path in $O(\sqrt{n})$ time (where n is the number of processors in the arm configuration map M_a). Limited space prevents us from describing the details of the path finding mechanism in this paper.

The two conditions necessary for path finding are that: (1) processors in M_a only correspond to arm configurations which are possible given the kinematics of the arm, and (2) processors which are neighbors in M_a correspond to arm configurations that are neighbors in the configuration space of the arm. The first condition is necessary in cases where the arm has restricted ranges of movement. The AAC must not try to move the arm into impossible configurations.

The second condition is necessary to ensure that a connected path of processors in M_a corresponds to a smooth and continuous movement of the arm. We now have a closer look at each layer and the laws that allow the AAC to find the necessary inter-layer connections and satisfy conditions for path finding.

3.2 Further Detail Regarding The Maps

We refer to the i^{th} processors in maps M_o, M_a and M_e as o_i, a_i and e_i respectively (see Figure 6). The activation values for these processors are denoted by $\alpha(o_i)$, $\alpha(a_i)$ and $\alpha(e_i)$, and the output signals by $\beta(o_i)$, $\beta(a_i)$ and $\beta(e_i)$. The input weight vectors from the joint angle sensors to processor a_i are denoted $\mu(\mathbf{a_i})$; the input weight vectors from the eye motor angle sensors to processor ei are denoted by $\mu(\mathbf{e_i})$. The inter-map connection weight from processor o_i to a_j is denoted $\omega(o_i,a_j)$. Similarly, the connection weight from processor e_k to a_j is denoted $\omega(e_k,a_j)$. The special **boundary processor b** which occurs in equation 3.2.4 is described in section 4.3.

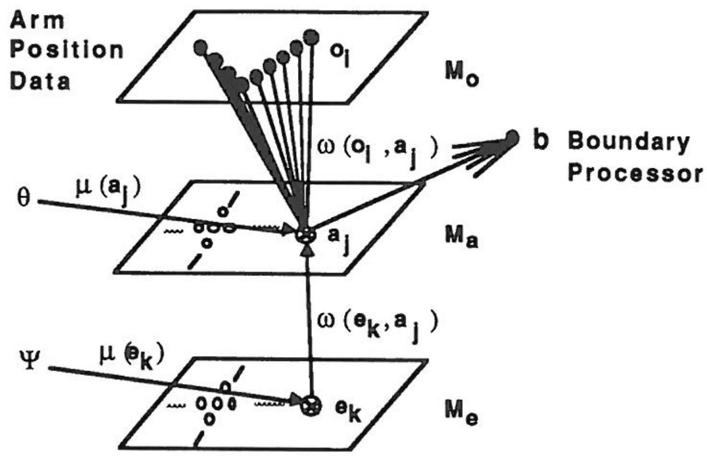


Figure 6: Learning Involves Weight Modification

The **obstacle map** is an egocentric image of the robot's workspace centered on an arbitrary fixed point (such as the first joint of the arm - see Figure 5a). Imagine a D-dimensional grid superimposed onto the workspace as in Figure 5b. Each grid cell is the receptive field of a corresponding M_o processor. This processor is active only if an obstacle falls into the receptive field. In a 2-dimensional workspace, the obstacle map activation pattern most naturally corresponds to a high-contrast, or back-lit image produced by a camera suspended above the robot. Equations 3.2.1 to 3.2.6 describe the post-training activation and output signal rules for processors in M_o, M_a and M_e.

POST-TRAINING PHASE:

$$\alpha(o_i)_t = \begin{cases} 1, & \text{if obstacle in receptive field} \\ 0, & \text{otherwise} \end{cases} \qquad [3.2.1]$$

$$\beta(o_i)_t = \alpha(o_i)_t \qquad [3.2.2]$$

$$\alpha(a_j)_t = \|\theta_t - \mu(\mathbf{a_j})_t\| \qquad [3.2.3]$$

$$\beta(a_j)_t = \begin{cases} 1, & \text{if } \alpha(a_j)_t = \min_{\forall q} \|\theta_t - \mu(\mathbf{a_q})_t\| & \text{"Winner" OR} \\ & \quad \beta(e_k)_t \cdot \omega(e_k,a_j) > 0 \text{ (and not Inhibited)} & \text{"Target"} \\ -1, & \text{if } \beta(o_i)_t \cdot \omega(o_i,a_j) > 0 \text{ OR } \beta(b)_t \cdot \omega(b,a_j) > 0 & \text{"Inhibited"} \\ 0 & \text{otherwise} & \text{"Inactive"} \end{cases} \qquad [3.2.4]$$

$$\alpha(e_k)_t = \|\theta_t - \mu(\mathbf{e_k})_t\| \qquad [3.2.5]$$

$$\beta(e_k)_t = \begin{cases} 1, & \text{if } \alpha(e_k)_t = \min_{\forall q} \|\theta_t - \mu(\mathbf{e_q})_t\| \\ 0, & \text{otherwise} \end{cases} \qquad [3.2.6]$$

The processors in the **arm** and **eye configuration maps** of the AAC receive configuration vectors as input from the robot. Each processor has a weight vector μ associated with its input lines. When an input is presented to the network, processors compete to "represent" it. The processor with the weight vector closest to the the input vector in the input space wins the competition by becoming active. This behavior quantizes the input space into Voronoi regions which represent sets of input vectors that activate each processor. The precision of the system is directly related to the resolution of this Voronoi tessellation as shown in Graf[6], and can therefore be made as fine as desired by adding processors to the system. Equations relating the number of nodes per processor to the allowable error are also given there.

These maps also have self-organizing properties that satisfy the two conditions described in section 3.1. Both conditions are necessary in Ma to ensure error-free path finding. Any self-organizing laws that (1) cause the distribution of weight vectors in the input space to asymptotically converge on an approximation of the training data distribution, and (2) preserve the topology of the network in the input space may be used (see Erdi and Barna[4], Kohonen[10,11], Ritter and Schulten[17]). We have employed the laws described by Kohonen[10] in our simulation. The first condition ensures that weights μ are distributed over the same area of the configuration space as the training configurations. Since arm configurations that occur during training are *by definition possible*, processors on a path through Ma (after training) will have weights that correspond to legal arm configurations. The second condition ensures that adjacent processors on a path will have weight vectors that are neighbors in configuration space and therefore represent only small changes in arm configuration. Such a path represents smooth and continuous movements of the arm. The M_a path finding mechanism must be capable of distinguishing between the winning (current) configuration processor (start of path), active target processor (end of path), inhibited configuration processor (not on path) and all other processors (possibly on path).

Finally, the M_o and M_e maps are fully connected to the M_a map; i.e., every processor in the obstacle and eye configuration maps is connected to every processor in the arm configuration map via a link weight ω. A weight of 0 indicates no connection; this is the initial situation for all processors. During training, if two interconnected processors are simultaneously active, the connection strength is increased, thus "growing" a connection between the processors; otherwise, the connection decays, eventually forgetting any association between the processors.

4. How The System Learns

In the simplest version of the system we have simulated, only the arm is represented by activations in M_o during the training phase and obstacles are not introduced until training is complete. Before training begins, all μ are set to random values and all ω are set to 0. As the arm is driven by the RMG, the end effector randomly wanders through a representative set of workspace points. The EET tracks this movement with the eyes. Sensors are sampled at regular intervals to (1) activate M_o processors with receptive fields intersected by the arm, (2) provide M_a with a joint angle vector θ, and (3) provide M_e with an eye angle vector Ψ. Although it is possible to adapt both μ and ω weights simultaneously, a two step training cycle has proven to be quicker and more accurate.

4.1 Self-Organization Step

Kinematic constraints on arm movement and viewable workspace points are represented by the shapes of the distributions of training θ and Ψ respectively. During this step, the self-organizing laws described in section 3.2 distribute M_a weights $\mu(\mathbf{ai})$ over the region of the arm configuration space that corresponds to legal movements of the arm and M_e weight vectors $\mu(\mathbf{ei})$ are distributed over the region of eye configuration space that corresponds to eye angles that focus the eyes on points of gaze in the workspace of the robot. Furthermore, the M_a and M_e network topologies are preserved in these spaces. Equations 4.1.1 to 4.1.8 express the self-organizing rules used during training.

TRAINING PHASE:

$$\alpha(o_i)_t = \begin{cases} 1, & \text{if arm in receptive field} \\ 0, & \text{otherwise} \end{cases} \quad [4.1.1]$$

$$\beta(o_i)_t = \alpha(o_i)_t \quad [4.1.2]$$

$$\alpha(a_j)_t = \|\theta_t - \mu(\mathbf{a_j})_t\| \quad [4.1.3]$$

$$\beta(a_j)_t = \begin{cases} 1, & \text{if } \alpha(a_j)_t = \min_{\forall q} \|\theta_t - \mu(\mathbf{a_q})_t\| \\ 0, & \text{otherwise} \end{cases} \quad [4.1.4]$$

$$\mu(\mathbf{a_j})_{t+1} = \mu(\mathbf{a_j})_t + \lambda_t \cdot \delta(a_j)_t \cdot (\theta_t - \mu(\mathbf{a_j})_t) \quad [4.1.5]$$

$$\alpha(e_k)_t = \|\theta_t - \mu(\mathbf{e_k})_t\| \quad [4.1.6]$$

$$\beta(e_k)_t = \begin{cases} 1, & \text{if } \alpha(e_k)_t = \min_{\forall q} \|\Psi_t - \mu(\mathbf{e_q})_t\| \\ 0, & \text{otherwise} \end{cases} \quad [4.1.7]$$

$$\mu(\mathbf{e_k})_{t+1} = \mu(\mathbf{e_k})_t + \lambda_t \cdot \delta(e_k)_t \cdot (\Psi_t - \mu(\mathbf{e_k})_t) \quad [4.1.8]$$

λ_t is a function which exponentially decreases from 0.5 to 0.01 during the first 3,000 iterations of the self-organizing step. $\delta(p)_t$ is a function with value 1 if processor p falls within a radius of r processors from the winning (active) processor. Radius r decreases from 0.25 times the width of the network to 1 during the first 3,000 iterations. We have found that good approximations to the training distributions are typically achieved within 60,000 iterations.

4.2 The Inter-Map Association Step

Eye/hand coordination constraints are represented in the training data by consistencies between θ and Ψ vectors. Obstacle avoidance constraints are represented by consistencies between M_o activation patterns and θ vectors. During this step equation 4.2.1 causes connections to grow between each M_o processor o_i and all M_a processors a_j corresponding to arm configurations that intersect the receptive field of o_i. These inhibitory connections act as an adaptive data structure which eliminates impossible arm configurations from movement paths. Equation 4.2.2 describes connection growth between M_e processor e_k and all M_a processors a_j corresponding to arm configurations that position the end-effector at points of gaze represented by e_k. These excitatory connections record potential target configurations for eye configurations.

$$\omega(o_i,a_j)_{t+1} = \omega(o_i,a_j)_t + \xi\cdot\beta(o_i)_t\cdot\beta(a_j)_t - \chi\cdot\omega(o_i,a_j)_t \qquad [4.2.1]$$

$$\omega(e_k,a_j)_{t+1} = \omega(e_k,a_j)_t + \xi\cdot\beta(e_k)_t\cdot\beta(a_j)_t - \chi\cdot\omega(e_k,a_j)_t \qquad [4.2.2]$$

Coefficients ξ and χ control the rate of connection growth and decay. Similar equations have been extensively studied by Grossberg (e.g. Grossberg[7]). The density of training points needed is dependent on the number of Voronoi regions in M_a and M_e. We have found that approximately 10,000 iterations are sufficient for networks of 900 processors or less.

4.3 Boundary Determination

Paths through M_a must not cross regions of configuration space corresponding to impossible arm movements. A simple mechanism is used to inhibit processors on the boundary of such regions. Boundary inhibition guarantees that the path finder will generate paths which circumvent these regions rather than crossing them. Boundary processor b is active during training whenever any arm joint reaches a limit of movement, or the joint motor sensors detect resistance to movement (due to the body of the robot - see equation 4.3.1). Equation 4.3.3 describes inhibitory connection growth (during training) from b to all M_a processors that represent arm configurations bordering these regions. After training, processor b is permanently activated to inhibit bordering configurations.

TRAINING PHASE:

$$\alpha(b)_t = \begin{cases} 1, & \text{if (a joint is at limit of movement range) OR (resistance to movement occurs)} \\ 0, & \text{otherwise} \end{cases} \qquad [4.3.1]$$

$$\beta(b)_t = \alpha(b)_t \qquad [4.3.2]$$

$$\omega(b,a_j)_{t+1} = \omega(b,a_j)_t + \xi\cdot\beta(b)_t\cdot\beta(a_j)_t - \chi\cdot\omega(b,a_j)_t \qquad [4.3.3]$$

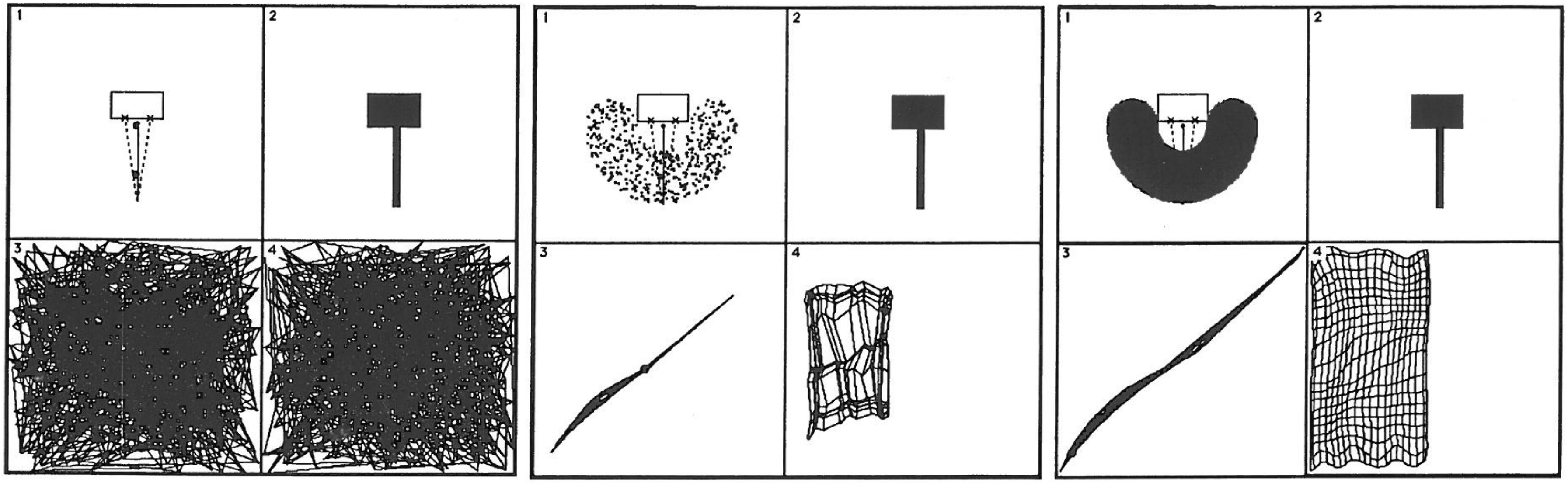
POST-TRAINING PHASE:

$$\alpha(b)_t = 1 \qquad [4.3.4]$$

$$\beta(b)_t = \alpha(b)_t \qquad [4.3.5]$$

4.4 Ongoing Adaptation

Once initial learning is complete, unplanned kinematic and sensor calibration changes are accommodated by a process of ongoing adaptation. We provide only a simplified description in this paper. As the arm moves through paths selected by the AAC, the EEC tracks the end effector, generating new training data. This data is used to gradually alter the distributions of μ in M_a and M_e, and to grow new inter-map connections if the robot has changed in some way. If the changes are small and gradual, small shifts in μ will introduce only a few errors in inter-map connections. These errors will be corrected as these connections decay. Inter-map connections that remain correct will continue to be reinforced by training data. As expected, large, sudden changes will introduce more serious errors until the adaptive laws "catch up". In this case, it is more efficient to temporarily return to the initial training mode until the changes have been accommodated. Once a collision is detected, the decay rate χ can be increased to forget incorrect connections more quickly.

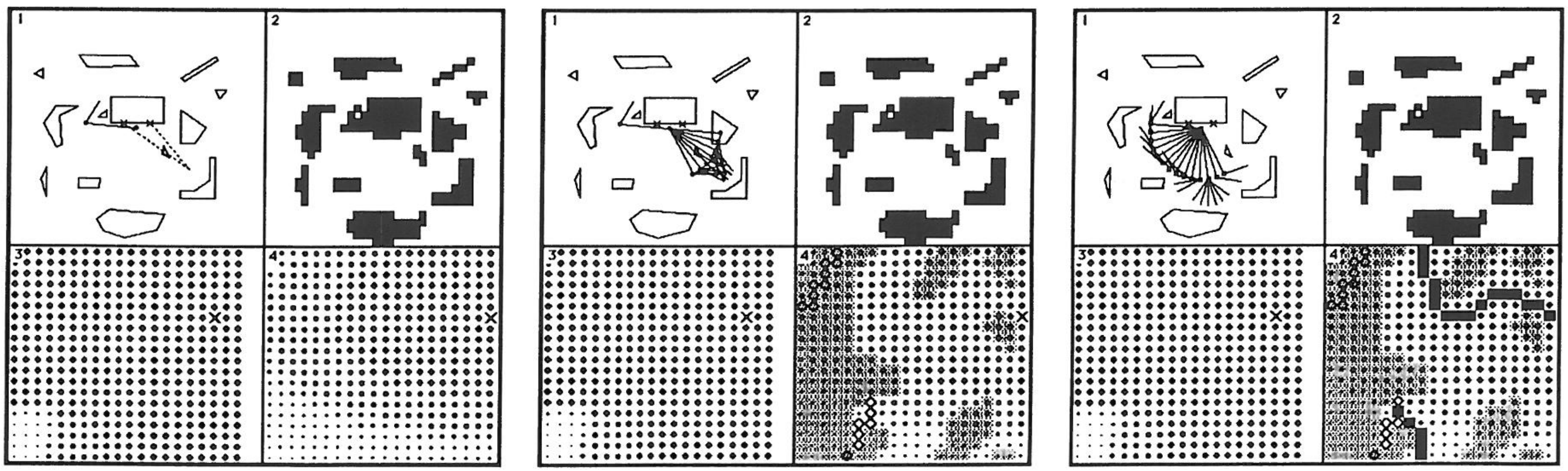


(a) 0 Iterations

(b) 500 Iterations

(c) 60000 Iterations

Figure 7: Example Training Run



(a) Direct eyes to target

(b) Find legal target configurations

(c) Select path and move

Figure 8: Generating an Arm Movement

5. Simulation Results

The results of a typical training run for a simple 2 degree-of-freedom robot are shown in Figure 7. The simulation is written in ParcPlace™ Smalltalk-80[1] and runs on a Sun workstation. Figure 7a depicts the four simulation window views (numbered 1 through 4). View 1 is a representation of the robot and its workspace. The eyes are represented as two x's on the front of the body (rectangular in this case). The dotted lines extending from the eyes to the tip of the arm represent the lines-of-sight of each eye. View 2 represents the pattern of M_o activations (black squares are active processors). View 4 represents θ space. The horizontal axis represents the range of J_1 angles (0 to 2π) and the vertical axis represents the range of J_2 angles (0 to 2π). M_a weights are shown mapped into the space by connecting neighboring weights by line segments (as in Kohonen[10] and Ritter and Schulten[17]). View 3 similarly depicts the Ψ eye angle space and M_e weights. The horizontal axis represents C_1 angles (0 to 2π) and the vertical axis represents C_2 angles (0 to 2π). The sequence of windows 7a,b,c show the weight vectors μ at 0 iterations, 500 iterations and 60,000 iterations respectively. The M_a and M_e network weights gradually assume the same distribution as the training data. Each dot in view 1 represents the end-effector position of a training sample. Inter-map connections are not shown. Figure 8 shows the movement sequence generated for a target point of gaze. Obstacles have now been introduced into the workspace. The bottom views have been replaced with representations of the M_a and M_e networks. Each processor is represented by the distance of its weight vector μ to the input vector. Large dots represent proximal weights, small dots represent distant weights. The processors which win the input competitions are marked with X's, target processors are diamonds, inhibited processors are shaded gray and processors on the path are black rectangles. In Figure 8a the robot has directed the eyes to a target point of gaze. Processors representing the current eye and arm configurations are activated. Figure 8b shows the M_a network at the next instant in time. The winning M_e processor (view 3) has activated a set of 16 target arm configurations in Ma (view 4). Shaded processors in M_a have been inhibited by active processors in M_o. Notice that 10 target configurations have been inhibited in M_a (gray diamonds). These configurations intersect the two obstacles near the point of gaze. Figure 8c shows one of the many movement paths possible. The path generated here illustrates that paths may wrap around joint angle space if joints have unrestricted movement. The path begins at the current arm configuration (right side of view 4), extends up to $\theta_2 = 2\pi$ and then continues from $\theta_2 = 0$ (bottom of view) to a target configuration. The sequence of arm moves generated by this path are shown in view 1. Notice that L_2 first swings up and then down to avoid collisions with obstacles as it moves to touch the target point.

6. Conclusions and Future Work

We have presented a new approach for the control of general robot manipulators that possesses eight important characteristics. First, the approach is in principle capable of autonomously learning the kinematics and workspace constraints for any general robot manipulator, dispensing with the need for specialized kinematic programming. Second, the system autonomously learns eye-hand co-ordination, providing a natural form of task level control in which a target point is selected by "glancing" at it with visual sensors. Third, the system learns collision avoidance constraints which greatly simplify the task of collision-free path planning in an arbitrarily cluttered workspace. These constraints are independent of specific obstacle sets. Fourth, no additional learning is required when the obstacle set is changed. Fifth, the system can be made to adapt to changes in arm-body kinematics and joint sensor calibrations, reducing the need for maintenance. Sixth, the system can be made sensitive to the domain specific distribution of arm movements; i.e., it can automatically increase the accuracy of the arm in those regions of the workspace where precision is most needed. Seventh, the system is tolerant of random noise in the training data. Finally, the approach is fast since it is most naturally implemented by massively parallel hardware.

This approach improves upon previous adaptive arm control schemes in five ways: (1) collision avoidance constraints that guarantee collision-free movement for all possible obstacle sets are learned once, (2) more than one target arm configuration is proposed for a given point of gaze generalizing the work of Kuperstein[12], (3) "virtual" obstacles as used, for example, in Tsutsumi[18] are not required to avoid collisions, (4) the system handles target points outside of the robot's workspace in a natural manner — the arm simply reaches toward these points, and (5) the system can be made arbitrarily precise by increasing the number of network processors. The primary disadvantage of this system is the rapid increase in the number of processors needed to provide high degrees of positioning accuracy. Optical technologies may hold the greatest promise for providing the required density of neurons and connections.

In order to enhance the practicality of our approach, we are currently investigating techniques which reduce the number of processors and connections needed for arms of many degrees of freedom.

Acknowledgement

This research has been supported by NSERC, the National Science and Engineering Research Council of Canada.

---

[1] ParcPlace Smalltalk is a trademark of ParcPlace Systems Inc.

References

1. Abu-Mostafa, Y.S., Psaltis, D., Optical Neural Computers, Scientific American, 256(3), 1987, pp. 88-95.
2. Amari, S., Topographic Organization of Nerve Fields, Bull. Math. Biol. 42, 1980, pp. 339-364.
3. Bullock, D., Grossberg, S., A Neural Network Architecture for Automatic Trajectory Formation and Coordination of Multiple Effectors During Variable-Speed Arm Movements, Proc. IEEE First Intl. Conf. Neural Networks, San Diego, Cal., June 1987, vol. 4, pp. 559-566.
4. Erdi, P., Barna, G., Self-Organizing Mechanism for the Formation of Ordered Neural Mappings, Biol. Cybern. 51, 1984, pp. 93-101
5. Faverjon, B., Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator, Proc. IEEE Int. Conf. on Robotics, Atlanta, GA, March 1984.
6. Graf, D.H., A Neural Controller For Collision-Free Movement of Robot Manipulators, School of Computer Science MCS thesis, Carleton University, September, 1988.
7. Grossberg, S., Classical and Instrumental Learning by Neural Networks, in *Progress in Theoretical Biology*, Rosen and Snell Eds., 1974, pp. 51-141.
8. Grossberg, S., Kuperstein, M., *Neural Dynamics of Adaptive Sensory-Motor Control*, North-Holland, 1986.
9. Hecht-Nielson, R., Counterpropagation Networks, Proc. IEEE First Intl. Conf. Neural Networks, San Diego, Cal., June 1987.
10. Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, 1984.
11. Kohonen, T., Learning Vector Quantization for Pattern Recognition, Helsinki University of Technology, Report TKK-F-A601, November 1986.
12. Kuperstein, M., Adaptive Visual-Motor Coordination in Multijoint Robots Using Parallel Architecture, IEEE Intl. Conf. Robotics and Automation, Raleigh, NC, March 1987, pp. 1595-1602.
13. Laugier, C., Germain, F., An Adaptive Collision-Free Trajectory Planner, Proc. Int. Conf. Adv. Robotics, Tokyo, Japan, September 1985.
14. Lozano-Perez, T., A Simple Motion-Planning Algorithm for General Robot Manipulators, IEEE Journal of Robotics and Automation, Vol. RA-3, No. 3, June 1987, pp. 224-238.
15 Miller, R. and Stout, Q.F., Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer, IEEE Transactions on PAMI, vol. 7, no. 2, March 1985, pp. 216-228.
16. Psaltis, D., Wagner, K., Brady, D., Learning in Optical Neural Computers, Proc. IEEE First Intl. Conf. Neural Networks, San Diego, Cal., June 1987, vol. 3, pp. 549-556.
17. Ritter, H., Schulten, K., On the Stationary State of Kohonen's Self-Organizing Sensory Mapping, Biol. Cybern. 54, 1986, pp. 99-106.
18. Tsutsumi, K. and Matsumoto, H., Neural Computation and Learning Strategy for Manipulator Position Control, Proc. IEEE First Intl. Conf. Neural Networks, San Diego, Cal., June 1987, vol. 4, pp. 525-534.
19. Tsutsumi, K., Katayama, K. and Matsumoto, H., Neural Computation for Controlling the Configuration of a 2-Dimensional Truss Structure, Proc. IEEE Second Intl. Conf. Neural Networks, San Diego, Cal., June 1988, vol. 2, pp. 575-586.
20. Elsley, R.K., A Learning Architecture for Control Based on Backpropagation Neural Networks, Proc. IEEE Second Intl. Conf. Neural Networks, San Diego, Cal., June 1988, vol. 2, pp. 587-594.