A real-time learning neural robot controller*

P. Patrick van der Smagt†
Ben J. A. Kröse

University of Amsterdam
Department of Computer Systems
Kruislaan 403, 1098 SJ Amsterdam
THE NETHERLANDS
email: smagt@fwi.uva.nl

Abstract

A neurally based adaptive controller for a 6 degrees of freedom (DOF) robot manipulator with only rotary joints and a hand-held camera is described. The task of the system is to place the manipulator directly above an object that is observed by the camera (i.e., 2D hand-eye coordination). The requirement of adaptivity results in a system which does not make use of any inverse kinematics formulas or other detailed knowledge of the plant; instead, it should be self-supervising and adapt on-line.

The proposed neural system will directly translate the preprocessed sensory data to joint displacements. It controls the plant in a feedback loop. The robot arm may make a sequence of moves before the target is reached, when in the meantime the network learns from experience. The network is shown to adapt quickly (in only tens of trials) and form a correct mapping from input to output domain.

1 Introduction

Traditionally, when a robot manipulator controller receives sensory information based on which an arm move has to be made, it calculates the trajectory, the inverse kinematics, and the inverse dynamics from an internal representation of preprocessed sensory data (see figure 1). Also, traditional

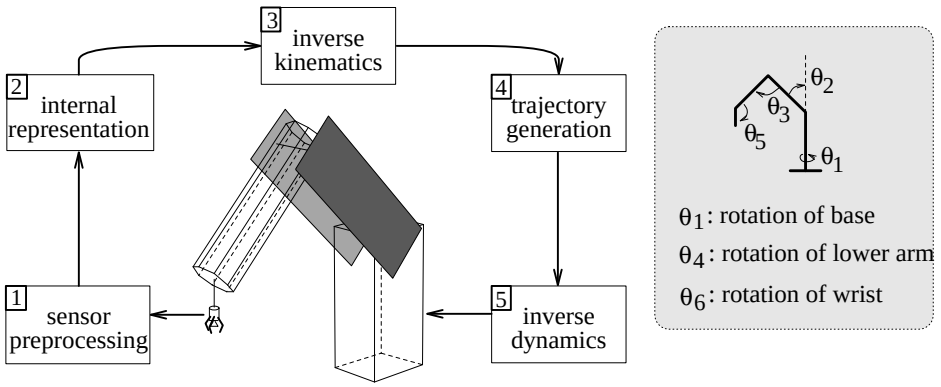

Figure 1: Traditionally, robot control involves several transitions from box 1 to box 5. The proposed neural system makes a direct transition from box 1 to box 5.

controllers are usually rigid, not allowing for changes in the sensors or manipulator (such as a rotated camera or a bent arm segment) without recalculating the whole system, which involves the intervention of an operator. Although non-neural adaptive controllers have been investigated, real-time computational requirements grow out of bound when the number of state variables increases [1].

*From: *Proceedings of the 1991 International Conference on Artificial Neural Networks*, ICANN–91, Espoo, Finland, June 24–28, 1991, pp. 351–356.

†This research has been partly sponsored by the Dutch Foundation for Neural Networks.

The introduction of neural networks in the field of robotics control adds to the ability to be adaptive (when no *a priori* knowledge of the system is used) and incorporate sensor data fusion.

A serious drawback with most neural controllers, however, is that they need tens of thousands of learning iterations before the systems exhibits acceptable behaviour. This poses no problem when the manipulator is simulated, but is unacceptable for real manipulators where a single move (together with sensory data processing) may take up to $0.5s$. The proposed neural controller differs from earlier approaches (e.g., [2–5]) in that it incorporates very fast training and retraining and is autonomously adaptive. It presents a direct transition from preprocessed sensory data to joint displacements (figure 1). The neural controller, manipulator, and sensory device constitute a feedback loop in which the manipulator is controlled.

2 Current approach

The hand of the (6 DOF) anthropomorphic robot arm is to be positioned above an object which is arbitrarily placed on the table. Although 3D information can be extracted using one hand-held camera we restricted ourselves to a 2D movement with the camera always looking down from a constant altitude. With these restrictions, only θ_1 and θ_3 are needed to control the manipulator while θ_2 and θ_5 can be expressed in terms of θ_1 and θ_3 (see figure 1). Both θ_4 and θ_6 are unused.

The proposed controller is based on a feed-forward network which translates a position in camera domain (i.e., the position of the observed object relative to the centre of the camera image) $\boldsymbol{x} = (x, y)$ plus the current state of the robot θ_3 to a joint displacement $\Delta\boldsymbol{\theta} = (\Delta\theta_1, \Delta\theta_3)$. The control-loop is feedback; during motion, the inverse dynamics controller accepts new setpoints.

The feed-forward network used consists of three inputs, ten hidden units with sigmoid activation function, and two output units. A bias input is provided to the hidden and output units. The network is trained with an implementation of the conjugate gradient optimisation technique with improvements as suggested by Powell [6]. This technique is described in the appendix. Since the conjugate gradient algorithm has to be performed over a number of training samples instead of one at a time, a set of training samples (the short term memory or STM) is built up and maintained at all times. When the size of the set exceeds n elements (typ. 100), the oldest element is replaced by the new one.

One cycle of the controller consists of the following operations. First, the current position of the object $\boldsymbol{x}$ and the current state of the robot θ_3 are measured and the STM is updated with the learning samples generated from the previous cycle. Next, $\boldsymbol{x}$ and θ_3 are fed into the neural network which generates a setpoint $\Delta\boldsymbol{\theta}$. This setpoint is sent to the plant, and during its move the network makes one learning iteration over the learning set. After this iteration, but not necessarily after the setpoint has been reached, the cycle is restarted. The whole procedure is repeated until the target position is reached[1].

New learning samples are constructed as follows. With the input $(\boldsymbol{x}, \theta_3)$ the network gives a joint angle displacement $\Delta\boldsymbol{\theta}$, resulting in a new position $\boldsymbol{x}'$. It can be shown [7] that the position that would have been reached with this $\Delta\boldsymbol{\theta}$ equals $\boldsymbol{x} - \boldsymbol{T}\boldsymbol{x}'$, with $\boldsymbol{T}$ the rotation matrix of the second camera image with respect to the first image. $\boldsymbol{T}$ can be deduced from the rotation $\Delta\theta_1$ or the rotation of the object as seen by the camera. The learning sample, which consists of $\Delta\boldsymbol{\theta}$, the corresponding displacement $\boldsymbol{x} - \boldsymbol{T}\boldsymbol{x}'$, and the state θ_3, is added to the STM. Also, a second sample is constructed from the summed moves from the start position and the corresponding total visual displacement.

The system is initialised with random weights (the long term memory or LTM) and four moves are loaded into the STM. These moves correspond to a right-up ($\Delta\boldsymbol{\theta} = \begin{pmatrix} +.1 \\ +.1 \end{pmatrix}$), left-up ($\Delta\boldsymbol{\theta} = \begin{pmatrix} -.1 \\ +.1 \end{pmatrix}$), right-down ($\Delta\boldsymbol{\theta} = \begin{pmatrix} +.1 \\ -.1 \end{pmatrix}$), and left-down ($\Delta\boldsymbol{\theta} = \begin{pmatrix} -.1 \\ -.1 \end{pmatrix}$) movement. Although the exact values are of no importance, four such learning samples are needed to prevent the following condition from

[1] A position is considered reached when the visual displacement (Euclidian distance) between the observed object and the centre of the camera image does not exceed 0.5 camera units or 1/20 of the visual field. This is roughly the attainable accuracy with the robot used and corresponds with the observed size of the object that is to be grabbed.

happening. When the system is initialised with random weights, it will usually output a joint displacement vector corresponding with a move in one direction only (such as left-up) independent of the network input. Since only this joint displacement is learned (together with the camera displacement it produces) the network will only reinforce a joint displacement in this direction. It will never learn to move in another direction. Initialising the STM with a displacement in every direction alleviates this problem.

2.1 Results

The performance of the controller can be measured as follows. An object is placed at a specific position within the visual field of the camera, while the manipulator is in its start position. The target position, together with the current value of θ_3, are fed into the neural network which produces a joint displacement $\boldsymbol{\theta}$. The manipulator moves in accordance with this displacement, and the resulting distance between the observed object and the centre of the camera image is measured. Repeating this experiment for several positions within the visual field of the camera produces the contour maps of figures 2, 4, and 5. The maximum displacement in both the x and y directions is ± 5.0 camera units (one camera unit corresponds with roughly 5 cm on the table), so the maximum error is $\sqrt{50} \approx 7.1$ camera units.

When starting from scratch, the controller typically makes 20 cycles before the first target position is reached. Figure 2 shows how the accuracy evolves during learning whilst trying to grab a

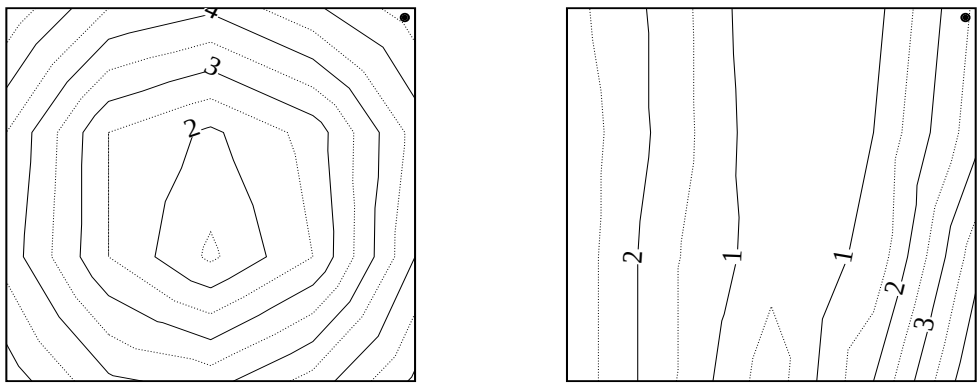

Figure 2: The error surface after 5 moves towards the target position (left) and after 10 moves (right) measured in camera units. The target position is situated at the upper right corner of each box and is marked by a black dot.

target object which is located in the upper-right corner of the camera image. After five cycles the manipulator still makes a movement independent of the target location, resulting in the high positional errors shown. After ten cycles positional errors for the main part of the input field are smaller than the eccentricity of the object, indicating that the arm is moving in the correct direction.

The path that is followed by the manipulator to reach the target position is depicted in figure 3.

After 15 cycles the target position has been reached for the first time and the error surface looks as depicted in figure 4 (left). In a second trial from the "home" position the target is reached after four moves, while the error surface improves even further (figure 4, right). Due to the fact that the initial weights used in this experiment made the manipulator initially move towards the upper left region of the picture, positional errors are lowest in this region. However, when the target has been reached twice, the error surface decreases in the region where the target position is situated.

Although positional errors remain relatively large far from the starting point, every position can be reached within the desired accuracy after two moves.

To illustrate the adaptive capabilities of the system, the camera is rotated by 45° and the task is repeated once by moving towards the position indicated by the black dot. The initial and final error surfaces are depicted in figure 5. The target is reached after 26 cycles in the first trial, and three moves in the second. These figures show the system's potential for adaptivity. Similar behaviour is exhibited when the camera is turned 90°, such that in effect the x and y inputs of the network are switched.

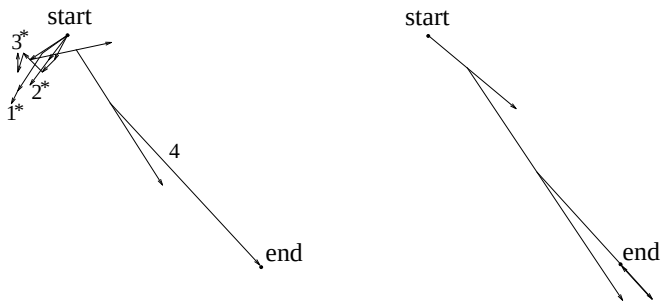

Figure 3: The paths followed by the manipulator when the target position is reached the first (left) and second (right) time. When a move has been partly executed a new setpoint is sent to the manipulator. In the left figure, the manipulator is initially moved in the wrong direction such that the target falls outside the visual field (indicated by an asterisk); the arm is set in the start position and the procedure is repeated.

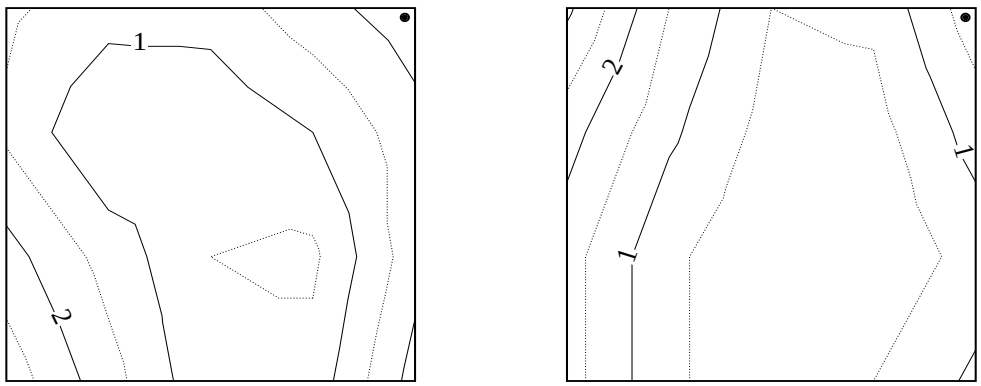

Figure 4: The error surface after the target position has been reached once (left) and twice (right).

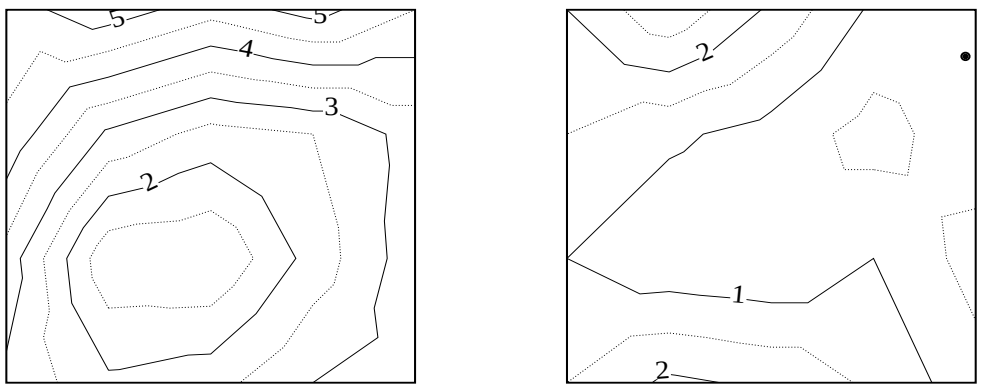

Figure 5: The error surface after a camera turn of $45°$ (left) and after the target point has been reached (right).

A learning curve is depicted in figure 6.

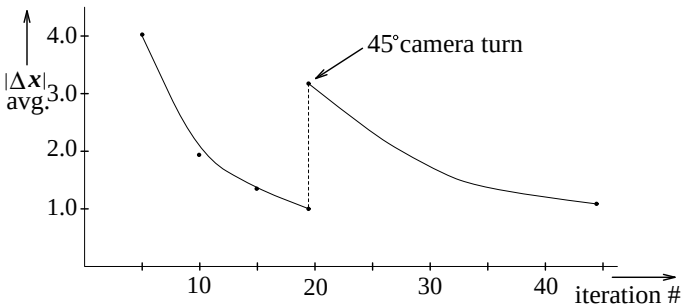

Figure 6: Error curve for the robot controller. On the vertical axis the average positional error after one move is plotted.

Appendix: The conjugate gradient algorithm

The proposed algorithm is an improvement on the error back-propagation algorithm for supervised learning of a feed-forward network. The algorithm is based on conjugate gradient optimisation [6, 8–12].

Conjugate gradient optimisation is a standard technique from numerical analysis. It replaces the primitive steepest descent method with a *direction set* minimisation method. Note that minimisation along a direction $\boldsymbol{u}$ brings the function f at a place where its gradient is perpendicular to $\boldsymbol{u}$ (otherwise minimisation along $\boldsymbol{u}$ is not complete). Instead of following the gradient at every step, a set of n directions is constructed which are all conjugate to each other such that minimisation along one of these directions $\boldsymbol{u}_j$ does not spoil the minimisation along one of the earlier directions $\boldsymbol{u}_i$, i.e., the directions are noninterfering. Thus one minimisation in the direction of $\boldsymbol{u}_i$ suffices, such that n minimisations in a system with n degrees of freedom bring this system to a minimum (provided the system is quadratic). This is different from gradient descent which directly minimises in the direction of the steepest descent. A more detailed discussion is provided in [13].

Although only n iterations are needed for a quadratic system with n degrees of freedom, due to the fact that we are not minimising quadratic systems, as well as a result of round off errors, the n directions have to be followed several times (see figure 7). Powell [6] introduced some improvements to correct for behaviour in non-quadratic systems. The resulting cost is $O(n)$ which is significantly better than the linear convergence[2] of steepest descent.

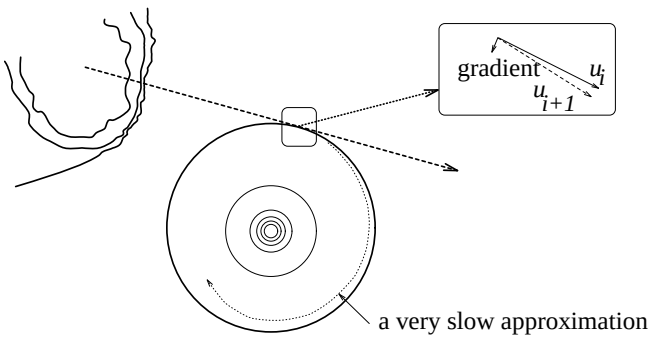

Figure 7: Slow decrease with conjugate gradient in non-quadratic systems.
The hills on the left are very steep, resulting in a large search vector $\boldsymbol{u}_i$. When the quadratic portion is entered the new search direction is constructed from the previous direction and the gradient, resulting in a spiraling minimisation [14]. This problem can be overcome by detecting such spiraling minimisations and restaring the algorithm with $\boldsymbol{u}_0 = -\nabla f$.

[2]A method is said to converge linearly if $E_{i+1} = cE_i$ with $c < 1$. Methods which converge with a higher power, i.e., $E_{i+1} = c(E_i)^m$ with $m > 1$ are called *superlinear*.

Example

As one example, figure 8 shows training curves for training a feed-forward network with two hidden units the XOR relationship. The improved training time is not only observed for such simple functions but for more complicated networks as well.

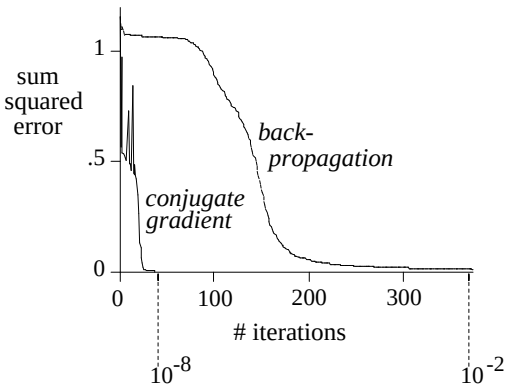

Figure 8: Error curve for conjugate gradient vs. steepest descent learning for an XOR network.

References

[1] A. H. Levis, S. I. Marcus, W. R. Perkins, P. Kokotovic, M. Athans, R. W. Brockett & A. S. Willsky, "Challenges to control: A collective view," *IEEE Transactions on Automatic Control* AC–32 (1987), 275–285.

[2] G. Josin, "Neural-space generalization of a topological transformation," *Biological Cybernetics* 59 (1988), 283–290.

[3] D. Psaltis, A. Sideris & A. A. Yamamura, "A multilayer neural network controller," *IEEE Control Systems Magazine* 8 (Apr., 1988), 17–21.

[4] H. J. Ritter, T. M. Martinetz & K. J. Schulten, "Topology-conserving maps for learning visuo-motor-coordination," *Neural Networks* 2 (1989), 159–168.

[5] M. Kuperstein, "INFANT neural controller for adaptive sensory-motor coordination," *Neural Networks* 4 (1991), 131–145.

[6] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical Programming* 12 (Apr., 1977), 241–254.

[7] B. J. A. Kröse, M. J. van der Korst & F. C. A. Groen, "Learning strategies for a vision based neural controller for a robot arm," in *IEEE International Workshop on Intelligent Motor Control*, O. Kaynak, ed., Istanbul, 20–22 Aug., 1990, 199–203.

[8] W. H. Press, B. P. Flannery, S. A. Teukolsky & W. T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986.

[9] J. Stoer & R. Bulirsch, *Introduction to Numerical Analysis*, Springer-Verlag, New York–Heidelberg–Berlin, 1980, *Einführung in die Numerische Mathematik*, Springer-Verlag, Heidelberg–Berlin, 1972, 1976.

[10] M. R. Hestenes & E. Stiefel, "Methods of conjugate gradients for solving linear systems," *Nat. Bur. Standards J. Res.* 49 (1952), 409–436.

[11] E. Polak, *Computational Methods in Optimization*, Academic Press, New York, 1971.

[12] E. Barnard & R. Cole, "A neural-net training program based on conjugate-gradient optimization," Oregon Graduate Center, Tech. rep. CSE 89–014, 1989.

[13] P. Patrick van der Smagt & B. J. A. Kröse, "A real-time learning robot controller," Dept. of Computer Systems, University of Amsterdam, internal report, Amsterdam, The Netherlands, (in preparation).

[14] Peter van Summeren, Sept., 1990, personal communication.