

# Neural Network Control of a Pneumatic Robot Arm

Ted Hesselroth<sup>†</sup> ,      Kakali Sarkar\* ,

P. Patrick van der Smagt<sup>†‡</sup>,

Klaus Schulten<sup>†</sup>

<sup>†</sup>Department of Physics and Beckman Institute

\*Department of Biophysics

University of Illinois

Urbana, Illinois 61801

<sup>‡</sup>Department of Mathematics and Computer Science

University of Amsterdam

Kruislaan 403, 1098 SJ Amsterdam

The Netherlands

April 28, 2005

## Abstract

A neural map algorithm has been employed to control a five-joint pneumatic robot arm and gripper through feedback from two video cameras. The pneumatically driven robot arm (SoftArm) employed in this investigation shares essential mechanical characteristics with skeletal muscle systems. To control the position of the arm, 200 neurons formed a network representing the three-dimensional workspace embedded in a four-dimensional system of coordinates from the two cameras, and learned a three-dimensional set of pressures corresponding to the end effector positions, as well as a set of  $3 \times 4$  Jacobian matrices for interpolating between these positions. The gripper orientation was achieved through adaptation of a  $1 \times 4$  Jacobian matrix for a fourth joint. Because of the properties of the rubber-tube actuators of the SoftArm, the position as a function of supplied pressure is nonlinear, nonseparable, and exhibits hysteresis. Nevertheless, through the neural network learning algorithm the position could be controlled to an accuracy of about one pixel ( $\sim 3$  mm) after two hundred learning steps and the orientation could be controlled to two pixels after eight hundred learning steps. This was achieved through employment of a linear correction algorithm using the Jacobian matrices mentioned above. Applications of repeated corrections in each positioning and grasping step leads to a very robust control algorithm since the Jacobians learned by the network have to satisfy the weak requirement that the Jacobian yields a reduction of the distance between gripper and target.

The neural network employed in the control of the SoftArm bears close analogies to a network which successfully models visual brain maps. We conclude, therefore, from this fact and from the close analogy between the SoftArm and natural muscle systems that the successful solution of the control problem has implications for biological visuo-motor control.

## 1 Introduction

With the advent of computers, the technique of extracting order from data by non-analytical means has begun to develop. The Kohonen neural network algorithm [1] is one method that has been shown to be effective for compressing a large data set into a representation by relatively few neurons while preserving the topological characteristics of the data set. This algorithm has been shown to work for both one-dimensional and multi-dimensional data sets [2, 3].

The vertebrate brain may employ a similar principle as that described by the Kohonen algorithm to order the data that it receives from sensory inputs [4, 5]. Simulations of the formation of visual maps were able to reproduce, on the basis of Kohonen-like network models, the organizational patterns of position, orientation, and ocular dominance as observed in the primary visual cortex of macaque monkeys [6, 7].

The Kohonen algorithm requires that the dimensionality of the data set be known a priori. It is desirable to have an algorithm which creates an internal representation of the data set which besides providing a compressed (vector

quantized) representation of the data set also models the data set's overall topology faithfully regardless of how each individual data point is represented. Such an algorithm, termed a manifold representing network (MRN) algorithm has been suggested recently [3, 8, 9] and will be employed in this study.

If processes similar to these algorithms are used in the brain to create structures which model the characteristics of input data, then performing experiments on artificial systems which use these algorithms may yield information on the brain's function. The study of a control system based on an NRM algorithm which learns to control the SoftArm is such an experiment, and may be useful from an engineering point of view as well. The algorithms employed are particularly applicable for all control systems which do not lend themselves to an analytical description, making design of a conventional control algorithm difficult.

The SoftArm robot arm shares essential characteristics with skeletal muscle systems. The arm uses actuators which consist of rubber tubes mounted on opposite sides of its rotating joints as presented in fig. 1. The arm has five joints. When air pressure is supplied to a tube, its diameter increases and its length decreases. The two tubes are connected by a chain across a sprocket, and when differing air pressures are supplied to the tubes, the differing equilibrium lengths result in a rotation of the joint. Thus, the arm operates on an agonist-antagonist principle just as in skeletal muscles; in particular, a set of pressures applied to the arm defines an equilibrium point for the arm posture. Furthermore, the posture is defined through pressure differences whereas the stiffness of the arm, i.e., the strength of the forces restoring the equilibrium posture, is determined by the sum of the pressures.

The need for a learning algorithm control of the SoftArm arises from the lack of an analytical relationship between arm pressure and arm posture. In addition, there is uncertainty in the position vs. pressure relation due to hysteresis in the response of the rubber tubes. These problems can be addressed by the use of feedback from the arm posture. While in conventional systems this feedback is usually in the form of joint angle information obtained from rotary encoders fixed to the joints, because of our interest in visuo-motor control we use visual feedback from the cameras during training and control of the robot.

An elementary task which is useful for demonstrating the learning algorithm for the robot arm is the task of positioning the end of the arm at a desired point in the workspace of the robot. To do this, the neural network must, when given the camera coordinates of the desired position, calculate the appropriate pressures to be supplied to the actuators of the arm. This is, therefore, a task of visuo-motor coordination; what is learned is the coordination of the visual receptive field with the mechanisms of motor control. Solution of the visuo-motor problem is part of our broader research agenda which is to understand how the brain connects reception and action.

A more advanced task is that of grasping. In this case the orientation of the object to be grasped must be taken into account as well as its position in Euclidean space. This increases the dimensionality of the problem, both for the image processing and for the joint control part of the tasks.

In our experiment, the tasks are learned and controlled by an NRM algorithm. We conceive of the information stored in a neuron's connections as being encoded 'in' the neuron. In each neuron is encoded a set of camera coordinates indicating position as well as orientation and a set of actuator pressures for the corresponding arm posture. If this were all the information encoded, then the number of possible positions would not exceed the number of neurons and only a very coarse representation of all possible tasks could be achieved. To obtain a finer representation the network interpolates between the distinct positions. This is done by also encoding a Jacobian matrix in each neuron and using it in a linear interpolation scheme, i.e., the neurons learn an affine map connecting camera coordinates (as indicated by lights affixed to the end of the robot arm) and joint pressures. Such a map is a local approximation to the true function relating the two.

This paper shows how all of the above information is learned from the camera and pressure information alone. The procedure used is based on previous work done in our laboratory [9, 10, 11, 12], including work on visuo-motor control of a conventional (PUMA 560) robot arm [13, 14]. It assigns neurons evenly to all postures of the robot arm as characterized through the visual field and teaches the neurons linear maps for guiding the arm to local targets.

## 2 The robot system

### 2.1 Robot system and environment

The robot arm was built mainly from components manufactured by Bridgestone Corporation of Tokyo, Japan. The whole system consists of the robot arm, an air compressor, servo-drive units and servo-valve units, the gripper, the cameras, and a host computer with vision system and serial interface boards. We will discuss these components below.

#### 2.1.1 The robot arm and its actuators

The robot is a four-link manipulator with five degrees of freedom. It is mounted by suspending it from its top joint (see fig. 1). The arrangement of the joints and their range of movement is basically modelled after the human arm. Because its pneumatic actuators, each consisting of two or four inflatable rubber tubes named *rubbertuators*, are relatively light, the arm weighs only 12 kg and can lift 3 kg. Because of its lightness and compliant characteristics, this arm can be employed for application around human operators or fragile equipment. Intended uses are in hospitals, around the handicapped, for household tasks and in areas where electrical circuits cannot be introduced. The dimensions and range of movement of the joints are given in Table 1. ignoretrue

The torque applied to each joint can be controlled by setting the pressures of the agonist-antagonist *rubbertuator* pairs. The *rubbertuators* are fixed parallel to each other in link  $i - 1$ . The free ends are connected to each other by a chain. The chain goes around a sprocket fixed in link  $i - 1$  and connected to link  $i$ .

The angular position of joint  $i$  thus depends on the relative lengths of the tubes as shown in fig. 3.

The joint angle  $\theta$  for each joint depends on the rubeertuator lengths  $l_1$  and  $l_2$  according to

$$\theta = \frac{l_1 - l_2}{2\pi r} \quad (1)$$

where  $r$  is the radius of the sprocket.

One of the greatest advantages of a rubeertuator is its very high force-to-weight ratio, about 240, compared to a value of about 16 for DC servo motors. This is especially good for robotics applications in which the actuators for the extreme joints are in motion as part of the arm.

The *stiffness* of any joint is controlled by means of the total pressure of the rubeertuators that drive it. When this total pressure is high, the joint behaves relatively stiffly, whereas a low pressure results in a compliant joint.

### 2.1.2 The air compressor and dryer units

The robot is supplied with compressed air of constant pressure throughout the experiment. First, the air is drawn from an in-house air compressor at about 90 psi. It then passes through a Balston 75-20 compressed air dryer (Balston Inc., Lexington, MA) in order to reduce moisture which would shorten the life of the robot. The subsequent dew point is  $-100^\circ$  F. Then it passes into a twelve gallon buffer tank. The purpose of the buffer tank is to even out any fluctuations in the supply pressure caused by sudden consumption by the actuators. It is then reduced to 75 psi by a pressure regulator. However, the robot is operated at less than maximum stiffness, so the highest pressure actually fed to the actuator is about 60 psi.

### 2.1.3 The servo-drive units

The servo-drive units (SDU's) provide the internal control circuitry for the robot. These units take signals from the host computer and operate the servo-valve units to obtain the proper response from the robot. There are five servo-drive units for the five actuators of the robot. The input for each unit is an RS-232 serial data terminal line. The units can receive either binary or ASCII-encoded signals and are operated at 9600 baud. The servo-drive unit operates digitally at 11 bits precision but converts its output to an analog signal. The output to the servo-valve unit is a pair of currents, one for each tube of the actuator, which are sent through shielded cable. The servo-valve units cause a pressure to be supplied to the tube which is proportional to the current. The current ranges from 4 mA for minimum pressure to 20 mA for maximum pressure.

### 2.1.4 The servo-valve units

The servo-valve unit (SVO) senses the pressure to each tube it controls and converts the pressure information to an electrical signal. The pressure may be

controlled by opening or closing electric valves. A standard control circuit is used to obtain a pressure which is proportional to the input current.

### 2.1.5 The gripper and its controlling valves

A gripper of about 1 kg is installed at the end of the arm. It has a simple two-fingered clamping action and is powered by air pressure. The fingers are approximately 10 cm long. Two inlets are required, one for opening and the other for closing. The air pressure is supplied through electric valves which can be controlled by the computer.

### 2.1.6 The video cameras

The cameras which are used for the position feedback to the neural network are two Cohu 4810 (Cohu Inc., San Diego, CA) monochrome CCD types with  $754 \times 488$  picture elements (pixels) producing a resolution of  $565 \times 350$  pixels. The cameras have 25 mm lenses and are located about 2 m from the robot with their lines of sight at approximately right angles.

### 2.1.7 The host computer

The host computer, a Sun 4-370, is equipped with two Androx ICS-400 parallel image array processors (Androx Corp., Canton, MA) which can do fast image processing. The position and orientation of the end-effector are extracted via two small lights fixed to the gripper of the robot, using the image array processors' abilities to find the brightest region of a frame and the center of that region.

## 2.2 Dynamics of the SoftArm

The servo-drive units allow the robot to be controlled in two modes: *position control mode* and *pressure control mode*. When the SoftArm is controlled in position control mode, an internal PID controller (see, e.g., [15]) is used in a feedback loop. This PID controller uses joint position feedback from the optical shaft encoders mounted on each joint to determine the pressure of the joints in a feedback loop. Figure 3 shows a representative move of one joint of the robot arm. The feedback mechanism should generate a smooth motion, but due to incorrect feedback signals the move is oscillatory. That the simple PID model is insufficient to control the robot arm will become evident from the following.

In pressure control mode, the pressure values sent by the host computer are directly translated to a current for the valves and the rubeator pressures are set correspondingly. The pressure generates a force in the rubeators which makes the joint rotate to assume a new equilibrium position.

### 2.2.1 Behaviour of a rubbertuator driven joint

Each actuator consists of a rubber tube sealed on one end and with an air inlet on the other end. The contraction force  $F_j$  exerted by rubbertuator  $j \in \{1, 2\}$  for each joint is specified by the manufacturer as

$$F_j = P_j D_j^2 (a(1 - \varepsilon_j)^2 - b) \quad (2)$$

where  $P_j$  is the supply pressure,  $a$  and  $b$  are constants depending on the particular tube,  $0 \leq \varepsilon_j < 0.2$  is the contraction ratio which is directly related to the rubbertuator length  $l_j$ , and  $D_j$  is the effective diameter of the tube before displacement. Although eq. (2) is not a precise model of the rubbertuators, it suffices to qualitatively explain their behaviour.

**Pressure–position relation.** By dividing both sides of eq. (2) by  $P_j$  it can be seen that for any specific choice of  $P_j$  there exists an infinite number of values  $\varepsilon_j$  and  $D_j$  which realise a specific exerted force  $F_j$ . Therefore, when a joint is in equilibrium, i.e., the external forces (gravity) are equal to  $F_1 - F_2$ , the joint angle is not only dependent on the pressure but also on the diameter of the tube before displacement. Since the diameter depends on the pressure and the elongation (before the displacement), the new joint position depends on the new pressure as well as on the previous position. This hysteresis can be shown by moving a joint along a pressure trajectory from  $P_1 = 0, P_2 = P_{\max}$  to  $P_1 = P_{\max}, P_2 = 0$  and back again by incrementing and decrementing the pressures by a constant value  $\Delta P$ . This results in the behaviour shown in fig. 4.

**Elasticity of the rubbertuators.** The long-term settling behaviour of the rubber has a large effect on the position of a joint *after* the desired pressure is reached and the joint seems to have reached its position. Figure 5 shows the position of joint 1 in time when the rubbertuators are allowed to settle for 200 seconds.

Influence of the temperature of the rubbertuators (which can occur due to varying climate conditions or simply by using the arm for extended periods of time) has also a large influence on the pressure–position relation. When repeatedly moving the robot to the same pressure, the system drifts gradually to different positions (see fig. 6).

From the above it is obvious that a precise model for the pneumatic actuators cannot be easily constructed. When the robot arm is used for accurate positioning and orientation of the end-effector, an adaptive algorithm is clearly necessary for controlling the robot.

## 3 Position control

### 3.1 Motivation of the Algorithm

It is believed that the brain constructs an internal representation of the visual field by a learning process [6]. While the mapping of visual features onto the retina is the same for each individual, it has been shown experimentally [16, 17] that the mapping of features in the neocortex develops in a manner which depends on the visual experience of the individual, i.e., depends on the stimuli that are input to the visual receptors. The conclusion is that the brain must be using a learning algorithm which induces this development. We hypothesize that the development of motor coordination proceeds in a similar way. It is one of the goals of our research to propose candidate algorithms that might be used for such development, and to test them on the robot system.

The MRN algorithm for neural networks has the property of forming a representation of a data set which maintains the topological features found in the data. This is so because the connections between neurons are established in a manner which depends on the probability distribution of the data set [18, 9, 3, 8]. In this way, a discrete set of neurons and their connections can model a continuous set of points of the input space. In order to get an accurate representation, the number of input data points generally exceeds the number of neurons. Thus, the function of the algorithms we propose is precisely data compression. In control applications, an internal model of part of the external environment is formed by the neural network. From this model, planning of actions for desired consequences is derived. Below, we apply such a network to the control of a robot arm.

Other methods [19, 20, 21] have been used for visuo-motor control of robots. In section 3.3 we provide a comparison between our results and the results reported in [19, 20, 21]. For another work on neural network control of a Softarm robot without the use of visual feedback, see ref. [22], which employs a hierarchical neural system for trajectory control of a single joint of the robot arm.

### 3.2 The Algorithm Used in the Robot Experiments

In our application each data point of visual input  $\mathbf{u}$  is a 4-dimensional vector (two dimensions from each of the two cameras of the system) of the visual space  $V \subset \mathfrak{R}^4$ . We model the connections from the visual input to the neurons of the network by a 4-dimensional vector. Thus, each of the 200 neurons of the network is considered to have assigned to it a position,  $\mathbf{w}_k \in \mathfrak{R}^4$ . The positions of the neurons are adjusted according to the data input  $\mathbf{u}$  using

$$\mathbf{w}_k^{\text{new}} = \mathbf{w}_k^{\text{old}} + \gamma(r, t) \cdot (\mathbf{u} - \mathbf{w}_k^{\text{old}}). \quad (3)$$

$\gamma$  is an indirect function of the distance between  $\mathbf{u}$  and  $\mathbf{w}_k$ . Specifically, it is a function of  $r$ , which is the ‘closeness ranking’ of neuron  $k$ . The closeness ranking

is determined by a metric which must be defined in the input space. If  $k$  is the closest neuron to  $\mathbf{u}$ , its closeness ranking is  $r = 0$ , if it is the next closest, its closeness ranking is  $r = 1$ , etc.  $\gamma$  is a monotonically decreasing function of  $r$ , in our application an exponential.  $\gamma$  also decreases monotonically with update step  $t$ , such that corrections to the positions  $\mathbf{w}_k^{\text{old}}$  become smaller as the network gradually learns the representation of the data. Explicitly, we used

$$\gamma(r, t) = e^{-r/\sigma} e^{-\sqrt{t}/\gamma} \quad (4)$$

with  $\sigma = 5$  and  $\gamma = 9$ . A similar network was employed in computer simulations of robot visuo-motor control [9, 10, 11, 12] and in visuo-motor control of a Puma 560 robot [13]. A detailed mathematical presentation of the algorithm is provided in [8].

If the network is to control the position of the robot arm, each neuron must also contain information on the posture of the arm which corresponds to the position stored by the neuron. This is a vector of pressures,  $\mathbf{P} \in \mathfrak{R}^3$ , since the arm has three joints. Each component is the pressure to one of the tubes of the actuator for the corresponding joint. The pressure to the other tube is a constant minus this pressure. The stiffness of the robot arm is controlled by this constant, (the sum of the pressures of the two actuator tubes) which for the present study was set to a high value, e.g., 60 psi. Even such high pressures in the actuator tubes render the arm much more compliant than a non-pneumatic robot arm.

In order for the network to be able to position the end effector of the robot to any target point  $\mathbf{u}$  in its workspace, it is necessary to interpolate between pressure values stored in neurons surrounding the target point. This is achieved by means of an affine map for the desired pressure vectors  $\mathbf{P}(\mathbf{u})$

$$\mathbf{P}(\mathbf{u}) = \mathbf{P}_k + A_k \cdot (\mathbf{u} - \mathbf{w}_k) \quad (5)$$

where  $k$  is the label of the neuron that is closest to  $\mathbf{u}$ .

Here,  $A_k$  is a  $3 \times 4$  matrix which provides a linear correction of pressure in accordance with the deviation  $\mathbf{u} - \mathbf{w}_k$  between the target point  $\mathbf{u}$  and the location  $\mathbf{w}_k$  assigned to neuron  $k$ . In the above we assume that the vector  $\mathbf{P}_k$  can be chosen independently of the current arm posture such that it moves the end effector to point  $\mathbf{w}_k$ . Due to hysteresis effects discussed previously (Section 2), this is an approximation to the true behavior.

$A$  in eq. (5) is the Jacobian which results as the first term in the Taylor expansion of  $\mathbf{P}(\mathbf{u})$  about  $\mathbf{w}_k$ .  $A$  may change according to position and so each neuron must store a matrix  $A_k$ .

In practice we calculate  $\mathbf{P}(\mathbf{u})$  from not just one neuron, but we average the above expression over several neurons in the neighborhood of the  $S$  neurons closest to neuron  $k$ , i.e., we employ the pressure vector

$$\bar{\mathbf{P}}(\mathbf{u}) = \sum_{r=0}^S \alpha(r) \cdot (\mathbf{P}_{k(r)} + A_{k(r)} \cdot (\mathbf{u} - \mathbf{w}_{k(r)})), \quad (6)$$

where  $k(r)$  is the label of the neuron which has closeness rank  $r$ .  $\alpha(r)$  is a monotonically decreasing function of  $r$ , that is, the neurons closest to  $k$  have the greatest weight in the sum. In our case, we used  $\alpha(r) = e^{-r/5}$ . The value of  $S$  was 50. The averaging enables the use of previous learning by neighboring neurons to decrease the positioning error [12].

The neuron pressures are updated according to a formula similar to that used for the neuron positions

$$\mathbf{P}_k^{\text{new}} = \mathbf{P}_k^{\text{old}} + \gamma(r, t) \cdot A_k(\mathbf{u} - \mathbf{v}_i) \quad (7)$$

where  $\mathbf{v}_i$  is the position of the arm after being set to the pressures  $\bar{\mathbf{P}}(\mathbf{u})$  specified by eq. (6), which we term the ‘coarse movement’. The vector  $(\mathbf{u} - \mathbf{v}_i)$  is the error after that movement.

From this error a correction  $\Delta\mathbf{P}$ , referred to below as a ‘fine movement’, may be calculated to the set of pressures of eq. (5). As in (6), averaging over a neighborhood is used:

$$\Delta\mathbf{P}(\mathbf{u}) = \sum_{r=0}^S \alpha(r) \cdot (A_{k(r)} \cdot (\mathbf{u} - \mathbf{v}_i)). \quad (8)$$

The Jacobians are then adjusted according to

$$A_k^{\text{new}} = A_k^{\text{old}} + \epsilon e^{-r/\sigma} \cdot A_k^{\text{old}}(\mathbf{u} - \mathbf{v}_f) \Delta\mathbf{v}^T \|\Delta\mathbf{v}\|^{-2} \quad (9)$$

where  $\mathbf{v}_f$  is the position after a fine movement and  $\Delta\mathbf{v} = \mathbf{v}_f - \mathbf{v}_i$ . The correction term here is the correlation matrix between the change in supplied pressure and the change in position of the end effector, though averaging has been ignored in order to correct neuron  $k$  specifically. After many learning steps,  $A_k$  will represent a local, linear approximation of the relation between position and pressure [9, 10, 11, 12, 13, 8].

The correction (8) can be applied several times for a given target position, with the most recent  $\mathbf{v}_i$  used in (8). After each correction  $\Delta\mathbf{P}$  is implemented, the fine position  $\mathbf{v}_f$  is obtained. Thus the change in position,  $\Delta\mathbf{v}$ , which corresponds to the change in pressure,  $\Delta\mathbf{P}$ , is obtained. Then eq. (9) can be applied, and the Jacobians may be updated several times for each target position. For the parameter  $\epsilon$  a value of 0.1 was chosen. Using  $\epsilon = 0.1$  takes into account the fact that several fine movements are used, and the Jacobians are adjusted after each fine movement.

It is easy to see that if  $A_k^{\text{new}}$  were to be used in (5) instead of  $A_k$ , it would be equivalent to adding a term  $A_k^{\text{new}}(\mathbf{u} - \mathbf{v}_f)$  on the right hand side, so that the error would be approximately  $\mathbf{u} - \mathbf{v}_f$  instead of  $\mathbf{u} - \mathbf{v}_i$ .

Slightly more sophisticated versions of the above algorithms may be employed, which have the advantage of using more information via averaging over neighboring neurons, the information having been obtained by updates of those neurons in previous learning steps. Equation (7) may be replaced by

$$\mathbf{P}_k^{\text{new}} = \mathbf{P}_k^{\text{old}} + \gamma(r, t) \cdot (\bar{\mathbf{P}}(\mathbf{u}) - \mathbf{P}(\mathbf{u}) + A_k(\mathbf{u} - \mathbf{v}_i)). \quad (10)$$

The difference between (10) and (7) is that the correction to  $\mathbf{P}(\mathbf{u})$  due to averaging has been added.

Similarly, instead of eq. (9) one may use

$$A_k^{\text{new}} = A_k^{\text{old}} + \epsilon e^{-r/\sigma} \cdot (\Delta \mathbf{P} - A_k^{\text{old}} \Delta \mathbf{v}) \cdot \Delta \mathbf{v}^T \|\Delta \mathbf{v}\|^{-2} \quad (11)$$

If  $\epsilon e^{-r/\sigma}$  were equal to 1 one would have

$$A_k^{\text{new}} \Delta \mathbf{v} = \Delta \mathbf{P}. \quad (12)$$

Then since  $\Delta \mathbf{P}$  was the actual adjustment of the pressures, and  $\Delta \mathbf{v}$  the measured change in the position,  $A_k^{\text{new}}$  would agree maximally with the available information from that step. However, its accuracy would be assured only for vectors parallel to  $\Delta \mathbf{v}$ . Therefore a Jacobian which is accurate for all directions can be obtained only after corrections from several directions are executed. It would appear that several learning steps for each neuron are required. However, the use of averaging reduces the number of steps required considerably [12]. The advantage of averaging is possible because the network employed is topology-conserving and hence allows us to identify which neurons are to be considered as neighbors.

### 3.3 Results

The target positions were chosen by assigning the components of a pressure vector randomly from a given range. Physically, this resulted in a workspace which was approximately a cube of 750 mm per side. The target coordinates were acquired by supplying the randomly chosen pressure to the arm and by gathering the position information from the cameras when the arm had come to rest. Then the coarse and fine movements towards the target position were executed in a similar manner. Of course, at this point no knowledge of the pressure corresponding to the target position was used.

The learning was unsupervised; only the target position and the information obtained from the coarse and fine movements were used for updating the neural network. There was some oscillation of the arm due to its compliant characteristics and due to the fact that the pressure changes were executed as a step function. The execution of the target, coarse, and fine movements altogether took an average of about 30 seconds per learning step.

We set a tolerance of allowed error vs time as follows

$$\|\mathbf{u} - \mathbf{v}_f\|_{\max}^{-2} = 1.51 + 150e^{-t^2/5000} \quad (13)$$

and for each step  $t$  repeated fine movements of the arm until the error between the actual position and the target position was less than this quantity. A plot of the final error vs step number is shown in fig. 7. All the distances were measured in terms of camera image pixels. One pixel corresponds to about 3 mm for the camera positions we used, or about 3% of the arm's length.

A measure of how well the robot has learned is furnished by the number of fine movements needed in order to reach targets within a given tolerance. In fig. 8. a plot of the number of fine movements required vs the number of learning steps  $t$  is shown. The tolerance is that of fig. 7. It was observed that the average number of fine moves required for  $t > 250$  (when the tolerance is near 1.5) is approximately 2. This is probably optimal considering that the neurons were rather widely spaced and a linear approximation was employed for the position vs pressure relation. Furthermore, hysteresis of the action of the rubber tubes was profound (cf fig. 4) and limited the accuracy of the Jacobians. A histogram of the number of fine moves for steps 485 to 1484 is provided in fig. 9.

For the mature network ( $t > 300$ ), the maximum number of fine moves required was 9. In a typical run, this happened once per thousand targets (learning steps 485–1484). In these cases the fine movement repeatedly over-shot the target, an indication that the values of the Jacobian were too large. Occasionally, about twice per thousand steps, the calculated fine movement was repeatedly very small. The correction term for updating the Jacobians (11) was then also very small. To circumvent the latter problem, if the tolerance was not met within 5 fine movements, a small movement was supplied to the arm in the direction of the last fine movement and the fine movement loop continued.

The mentioned errors in the Jacobians could have been due to insufficient learning, or overcorrections to the Jacobians during the early learning stages, when the prefactor of the correction term in (11) is largest. We chose the limits of the workspace to avoid singular points, which prevent the learning of the Jacobians. Another source of error is drift in the characteristics of the robot over time. After leaving the robot motionless for one hour, the run was restarted at  $t = 1000$  with the network values from the  $t = 1000$  timestep. Since  $t \geq 1000$  the learning has effectively been switched off. The choice of targets from  $t = 1000$  onward was also the same as in the previous run and so the results should have been identical. However, the average number of fine movements went from two to ten. It is obvious from the increase in the number of fine movements that the Jacobians from the first part of the run were no longer optimal. This was probably due to a cooling and stiffening of the rubber tubes of the actuators because a source of heating was removed during the one-hour pause, namely, internal friction in the rubber. Changes in the air temperature of the room, affecting the elasticity of the rubber tubes, can also be a factor in long runs.

Our results are comparable to those obtained with back-propagation methods on conventional robots. Cooperstock, *et al.* [21] obtain an accuracy of about 4% of the length of their robot arm after 63 steps using two fine movements, while our accuracy was about 3%. Kuperstein, *et al.* [19, 20] report an accuracy of 3% after 1200 steps for the coarse movement, and no detectable error with from two to four fine movements. However, in both of these methods the joint angles of the target position are used in the updating of the network values, whereas in our case, for the sake of biological realism, they are not known and the network has access to only those pressures generated by itself. In previous work [13] done by us on a conventional robot with the same algorithms described earlier (using joint angles instead of joint pressures), we obtained an accuracy

of about 2% of the robot arm length after 3000 learning steps with one fine movement.

## 4 The grasping task

Most of the nontrivial robotics applications demand control of the position as well as of the orientation of the end-effector. One of the basic capabilities of the end-effector is to be able to grasp objects. In this section, we will show how the positioning algorithm as presented in the previous section can be extended to incorporate the control of grasping movements.

### 4.1 Problem Description

Incorporation of the orientation control adds two dimensions to the positioning problem. The complete work space comprises a three-dimensional position space,  $R_x \subset \mathbb{R}^3$  and an embedded submanifold for orientation space,  $R_\theta \subset \mathbb{R}^2$ . Surrounding each position of the three-dimensional position space there exists a two-dimensional orientation space. Our goal is to generate finite discrete maps of these two spaces using the algorithm presented in section 3.2.

As mentioned earlier, the pneumatic robot used in our experiments has five degrees of freedom. The positioning control task made use of joints 1, 2 and 3. Joints 4 and 5 control the movement of the gripper.

Figure 10 shows a schematic diagram of the gripper mechanism. The gripper has two types of motions. The rotational motion about  $aa'$  is called *pitch* and it is a function of the sum of joint pressures  $P_4$  and  $P_5$ . The motion about  $bb'$ , called *roll*, depends on the difference between  $P_4$  and  $P_5$ . If either of joint pressure  $P_4$  or  $P_5$  is changed, both the abovementioned motions are generated. Accordingly, the two indicated motions of the gripper are mutually dependent.

As the gripper can rotate through  $180^\circ$  about  $bb'$ , while rotating about the  $aa'$  axis, it can span a two-dimensional space. However, it also produces a translational motion of the tip of the end-effector. Thus both the position and the orientation of the gripper are changed. This necessitates a unique combination of pressures in the five joints for each possible combination of the states of  $R_x$  and  $R_\theta$ . As the space  $R_x \otimes R_\theta$  will be considerably large, it would be preferable if we could represent each space in a separate network, i.e., separate the controls on  $R_x$  and  $R_\theta$ . This is not possible for two-dimensional orientation but can be achieved by constraining the orientation of the gripper to one dimension, e.g., allowing rotations only in a plane perpendicular to the axis of symmetry of the gripper. In other words, if the positioning is done in such a way that the plane normal to the plane containing  $aa'$  and  $bb'$  is parallel to the axis of symmetry of the cylinder to be grasped, a pure rotational motion about  $bb'$  will be sufficient to orient the gripper properly. Of course, the full grasping problem has not been solved then, since some orientations are excluded.

## 4.2 Network architecture

In order to map the five-dimensional space embedding  $R_x$  and  $R_\theta$  by a five-dimensional Kohonen net,  $N^5$  neurons are required where  $N$  is the number of elements of the Kohonen net along a single dimension. This will increase the search time for choosing the winner as  $O(N^5)$ . The space required for storing this network is also  $O(N^5)$ . In a previous attempt to reduce the search time a hierarchical Kohonen network was used [9]. This kind of network is characterized by a three-dimensional lattice of neurons for mapping the space  $R_x$ , each node of which consists of another two-dimensional layer of sub-lattices which map the space  $R_\theta$ . One of the advantages of this architecture is that the search time  $t_{\text{search}}$  increases only as

$$t_{\text{search}} = O(N_S^3 + N_E^2). \quad (14)$$

Here  $N_S^3$  is the number of two-dimensional sub-lattices used for mapping the space  $R_\theta$  and  $N_E^2$  is the number of nodes or elements in each of these sub-lattices. Here the storage space requirement is  $O(N^5)$  which would result if the five-dimensional space embedding  $R_x \otimes R_\theta$  would be mapped. The main disadvantage of using this type of network architecture is that it does not exploit the high degree of redundancy of the orientation control. In other words, there are many positions in  $R_x$  for which representations of the orientation of the gripper are nearly the same. Hence, it is unnecessary to map all these orientations separately for each set of subneurons. In order to exploit this redundancy we employed the following network.

Two sets of neurons are used for representing the input signals describing the location and the orientation of the object. One set stores the vectors and tensors,  $\mathbf{P}_k$  and  $\mathbf{A}_k$  associated with  $R_x$  and the other do the same for  $R_\theta$  (eq. (9)).

Two lights are mounted to the gripper on a line perpendicular to its symmetry axis. From any five different initial pressures at all the joints random pressures are applied to the first four joints and if the random pressure of the fourth joint is more than the initial pressure of that joint, the difference in them is subtracted from the initial pressure of the fifth joint and the resulting pressure is then applied to that particular joint. So the pressure of the fifth joint is increased or decreased by an amount equal to the computed decrease or increase in pressure of the fourth joint. This is done in order to obtain a pure rotational motion of the gripper around the  $bb'$  axis. After taking these pressures, the robot arm goes to a random position. Each camera records the positions of the lights, creating two sets of four-dimensional vectors. We define the position of the gripper as the mid-point of the two lights, and the orientation as the normalized vector between the lights. After taking this random position the robot arm moves to a new position and then tries to reach the target position and orientation again by means of the neural network. Since the positioning of the end-effector is independent of the orientation control, the Jacobian matrix,  $A_k$  (eqs. (10), (11)) for positioning is dependent only on the position information. Similarly the Jacobian corresponding to the orientation control depends only on orientational information. Thus the position and orientation control will be accomplished by two neural networks. This is equivalent to assuming complete

redundancy in the orientational information as encoded in the network.

Once the input signal is presented to the network, separate ranks of ‘closeness’ for both position and orientation are computed from the input signals. For each of these ranks, a simple four-dimensional Euclidean metric is used. Computation of the joint pressures are performed in a way similar to that described in section 3.2, with the only difference that the joint pressures for position control are computed from the  $R_x$  mapping and the joint pressures for orientation control are obtained from the  $R_\theta$  mapping. The search time in this architecture is  $O(N^3)$  but the storage requirement reduces to  $O(N^3)$  with  $O(N^3)$  neurons available for quantizing the space  $R_\theta$ . It is important to note that as we are not using the yaw motion here; our arm can execute only limited grasping movements.

### 4.3 Results

A network of 200 neurons has been used for mapping. Each neuron was associated with the weight vectors corresponding to the signals from  $R_x$  and  $R_\theta$ . Basically the same algorithm as described in section 3.2 was used for learning. For the results presented here, the  $e^{-\sqrt{t}/\gamma}$  term in the prefactor of eq. (4) has been replaced by  $e^{-t/\gamma}$ . Two Jacobian matrices of size  $3 \times 4$  and  $1 \times 4$  were stored for computing the positioning and orientation control pressures, respectively. A typical run demonstrates convergence of the network to one pixel error in positioning and an average of two pixel error in orientation after 800 learning steps, using  $\gamma = 100$ ,  $\sigma = 5$  and  $\epsilon_j = 0.1$ . Figure 11 shows for a typical run the dependency of the errors on the number of learning steps. Kuperstein, et al, included 2-DOF orientation in their work [19, 20] and reported orientation accuracy of  $60^\circ$  in solid angle. The orientation accuracy of the present work is approximately  $2^\circ$ .

## 5 Discussion

The SoftArm robot shows hysteretic behaviour and a pressure-position relationship which is strongly changing in time. In contrast with conventional robot systems, the SoftArm has not been designed to facilitate accurate posture control and, in fact, poses a challenging problem for adaptive control theory. The highly nonlinear and hysteretic behaviour of the arm obviously necessitates the use of adaptive algorithms for control.

We showed that it is possible to control the arm using pressure inputs alone. No explicit information about the joint angles is needed. Instead, feedback from the cameras is enough to control the position to an accuracy limited only by the cameras’ resolution.

Nonlinearity and hysteresis of the pressure dependence of joint position is significant for the SoftArm. Nevertheless, these characteristics did not seriously affect the performance of the algorithm. This is presumably because the effects are averaged out due to each neuron being trained over the course of many

learning steps. As seen from fig. 4, the derivative of position vs. pressure, and thus the correct values of the Jacobian, depend on the direction of motion of the joint. In the training session the Jacobians are trained over many steps, and the update information in eq. (11) can be acquired from either direction of motion. This results in an averaging of the values of the Jacobians over the correct, direction-dependent ones for each possible movement. The values of the Jacobians are thus approximate, but are close enough to the correct ones for each case so that the target can be reached after a few fine movements. We would like to stress here that the application of several fine movements does yield a robust control system: the Jacobians need not be known accurately; the only property required for the Jacobian is that its application leads to a reduction of the distance from the target.

The effect of the hysteresis is also diminished, somewhat artificially, by the fact that the arm posture previous to the coarse movement is always that of the target position, since the arm itself is used to acquire the target coordinates. This decreases the amount of travel performed by the arm in the coarse movement.

While we found it necessary to constrain the orientational space of the experiment to one dimension in order to control the orientation independently of position, one can view the full five-dimensional problem as consisting of the said one-dimensional orientation plus redundant four-dimensional positioning. In this way, all positions and orientations can be reached. The problem would then be reduced to choosing the posture appropriate for the desired orientation. The groundwork for this approach has been laid in ref. [9].

The value of the prefactor of the correction terms in eqs. (3), (10), and (11) is analogous to the plasticity of a natural neural network. We found that when the network is young, e.g., for low values of  $t$ , a high plasticity is desirable to quickly adjust the values stored in the neurons to approach the optimal ones, but that plasticity should then tail off to avoid losing information gained from earlier steps. If the plasticity remains too high, subsequent corrections to the values stored in the neurons effectively erases the values learned previously. We did not find it necessary to create a large neighborhood of influence in the network for early inputs, gradually decreasing the size of the neighborhood, as has been practiced in other applications [11, 13], but this could have been due to the initial random values of the network being not too different in overall magnitude from the correct values.

The overall time to reach a desired target is around 30 seconds. This time might be improved by using dynamical control, a possibility which is currently being investigated by us and others [23].

The research presented in this paper has shown the possibility of learning a non-linear, multi-dimensional function with a Kohonen neural network. This, along with other data [12, 6], implies their possible use in the brain's control systems, and also illustrates their potential for technological application.

## Acknowledgments

We would like to thank Volker Ehrlich and Benno Puetz for help with fig. 1, Joerg Walter for the vision system code, Tarcisio Campos for helpful discussions, and Frans Groen for reviewing the manuscript. The authors express their gratitude to the Carver Charitable Trust for support. Funds for the robot system were provided by the Beckman Institute through the Capital Development Board of the University of Illinois. The computations were carried out in the National Institutes for Health Resource for Concurrent Biological Computings (grant 1-P41-RR05969-01). This work has been partly sponsored by the Dutch Foundation for Neural Networks.

## References

- [1] T. Kohonen. Analysis of a simple self-organizing process. *Biol. Cybern.*, 44:135–140, 1982.
- [2] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, 43:59–69, 1982.
- [3] T. M. 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.
- [4] C. von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85–100, 1973.
- [5] D. J. Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. *Proc. R. Soc. Lond.*, B194:431–445, 1976.
- [6] K. Obermayer, G. G. Blasdel, and K. Schulten. A neural network model for the formation and for spatial structure of retinotopic maps, orientation- and ocular dominance columns. In T. Kohonen, editor, *Artificial Neural Networks*, pages 505–511. Elsevier Science Publishers (North Holland) Amsterdam, 1991. [Beckman Institute Technical Report TB-91-09].
- [7] K. Obermayer, K. Schulten, and G.G. Blasdel. A comparison between a neural network model for the formation of brain maps and experimental data. In D. S. Touretzky and R. Lippman, editors, *Advances in Neural Information Processing Systems 4*. Morgan Kaufmann Publishers, 1992. in press; [Beckman Institute Technical Report TB-92-01].
- [8] T. M. Martinetz and K. Schulten. Competitive hebbian rule forms manifold representing networks. *Neural Networks, submitted*.
- [9] Helge Ritter, Thomas Martinetz, and Klaus Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley, New York, 1992.
- [10] H. Ritter, T. Martinetz, and K. Schulten. Topology-conserving maps for learning visuomotor-coordination. *NN*, 2:159–168, 1989.
- [11] T. Martinetz, H. Ritter, and K. Schulten. Three-dimensional neural net for learning visuo-motor coordination of a robot arm. *IEEE Transactions on Neural Networks*, 1:131–136, 1990.
- [12] T. M. Martinetz and K. Schulten. A neural network for robot control: Cooperation between neurons causes learning. *Computers and Electrical Engineering, submitted to Special Issue on ‘Neural Networks: Theory and Applications in Robotics and Manufacturing’*.

- [13] J. A. Walter, T. M. Martinez, and K. Schulten. Industrial robot learns visuo-motor coordination by means of ‘neural-gas’ network. In *Proceedings of the International Conference on Artificial Neural Networks, Helsinki, 1991*. Elsevier Amsterdam, 1991.
- [14] J. A. Walter and K. Schulten. Implementation of a self-organizing neural network for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, 4:86–95, 1993.
- [15] John J. Craig. *Introduction to Robotics*. Addison-Wesley, Reading, MA, 1986.
- [16] G. Blasdel. Orientation selectivity, preference and continuity in monkey striate cortex. *J. Neurosci.* (submitted).
- [17] G. G. Blasdel and G. Salama. Voltage sensitive dyes reveal a modular organization in monkey striate cortex. *Nature*, 321:579–585, 1986.
- [18] H. Ritter and K. Schulten. On the stationary state of Kohonen’s self-organizing sensory mapping. *Biol. Cybern.*, 54:99–106, 1986.
- [19] M. Kuperstein and J. Rubinstein. Implementation of an adaptive neural controller for sensory-motor coordination. *IEEE Control Systems Magazine*, pages 25–30, April 1989.
- [20] M. Kuperstein. INFANT neural controller for adaptive sensory-motor coordination. *Neural Networks*, 4:131–145, 1991.
- [21] J. Cooperstock and E. Milios. Adaptive neural networks for vision-guided position control of a robot arm. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pages 397–403, Glasgow, Scotland, U.K., August 1992.
- [22] M. Katayama and M. Kawato. A parallel-hierarchical neural network model for motor control of a musculo-skeletal system. Technical Report TR-A-0145, ATR Auditory and Visual Perception Research Laboratories, April 1992.
- [23] M. Katayama and M. Kawato. Learning trajectory and force control of an artificial muscle arm by parallel-hierarchical neural network model. In *Proceedings of the IEEE Neural Information Processing Systems 1990*. Institute of Electrical and Electronics Engineers, 1990.

## Captions

Table 1. Dimensions of the links and motion range of the joints.

Figure 1. The robot system, showing SoftArm, air pressure supply, control electronics, and host computer.

Figure 3. An agonist and an antagonist rubeuator are connected via a chain across a sprocket; their relative lengths determine the joint position  $\theta_i$ .

Figure 4. Joint 1 of the robot arm as moved by applying a constant pressure increment  $\Delta P$  to rubeuator 1 and the same decrement to rubeuator 2. When the extreme pressures are reached, the direction is reversed.

Figure 5. Relaxation of joint 1 of the SoftArm in pressure control mode.

Figure 6. Drift of the rubeutors when the robot is used for a long period of time. The pressures of the rubeutors are repeatedly increased/decreased by 1% of the total pressure.

Figure 3. Joint 2 of the rubeuator robot moving in position control mode. Notice the jagged curve due to the feedback.

Figure 7. Final error between position and target, in pixels, vs. step number. One pixel corresponds to  $\sim 3$  mm.

Figure 8. Number of fine movements required to reach tolerance defined in eq. (13), vs. step number.

Figure 9. Histogram of number of fine movements required to reach the tolerance defined in eq. (13) for steps 485 to 1484.

Figure 10. A sketch of the gripper with two types of associated motions.

Figure 11. Positioning and orientation error a: positioning error; b: orientation error.

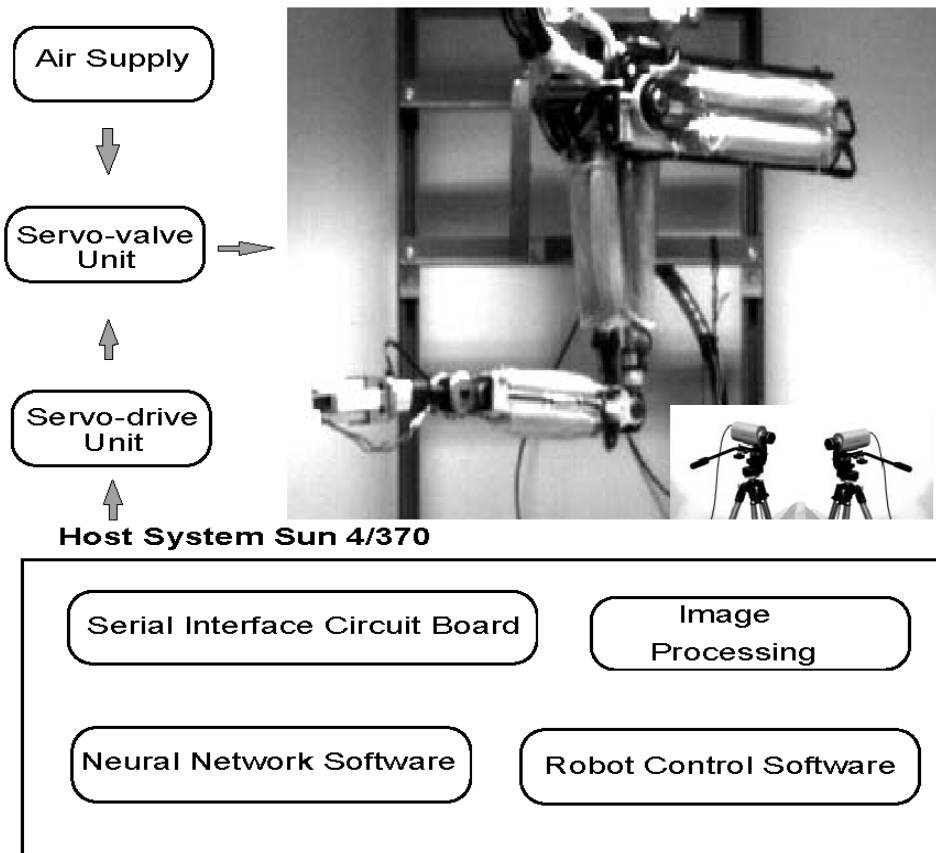


Figure 1: The robot system, showing SoftArm, air pressure supply, control electronics, and host computer.

Item			Specification
model			FAS-501
degree of freedom			5
rotation angle and arm length	first (shoulder)	angle length	$\pm 60^\circ$ –
	second (upper arm)	angle length	$\pm 50^\circ$ 410 mm
	third (lower arm)	angle length	$\pm 50^\circ$ 370 mm
	fourth (wrist pitch)	angle length	$\pm 45^\circ$ 270 mm
	fifth (wrist roll)	angle length	$\pm 90^\circ$ –
lifting capability			max. 3 kg

Table 1: Dimensions of the links and motion range of the joints.

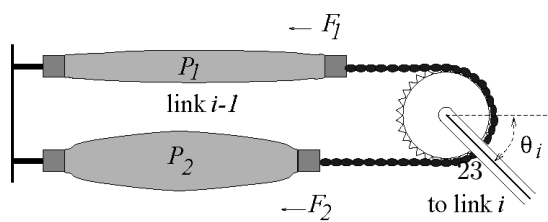


Figure 2: An agonist and an antagonist rubbertuator are connected via a chain across a sprocket; their relative lengths determine the joint position  $\theta_i$ .

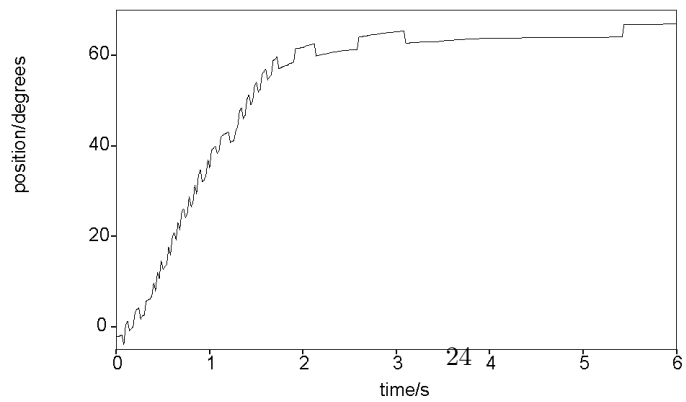


Figure 3: Joint 2 of the rubbertuator robot moving in position control mode. Notice the jagged curve due to the feedback.

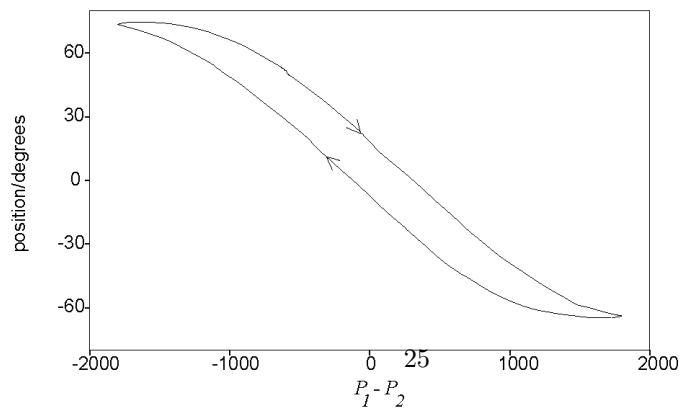


Figure 4: Joint 1 of the robot arm as moved by applying a constant pressure increment  $\Delta P$  to rubeuator 1 and the same decrement to rubeuator 2. When the extreme pressures are reached, the direction is reversed.

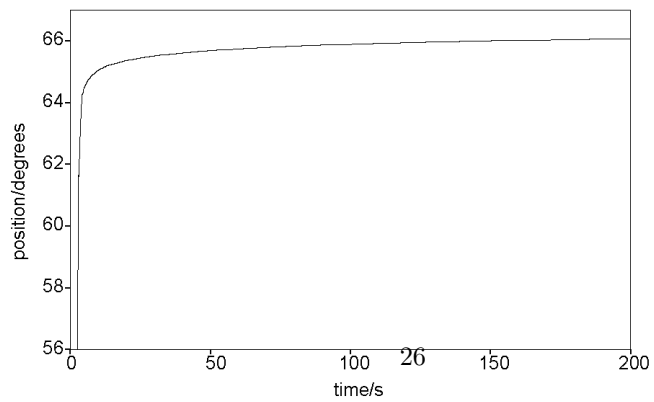


Figure 5: Relaxation of joint 1 of the SoftArm in pressure control mode.

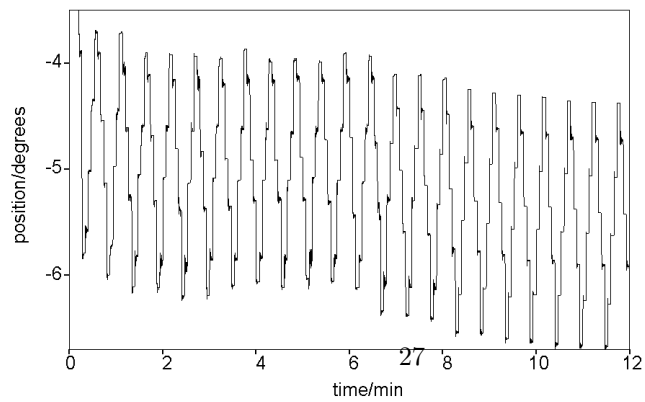


Figure 6: Drift of the rubeertuators when the robot is used for a long period of time. The pressures of the rubeertuators are repeatedly increased/decreased by 1% of the total pressure.

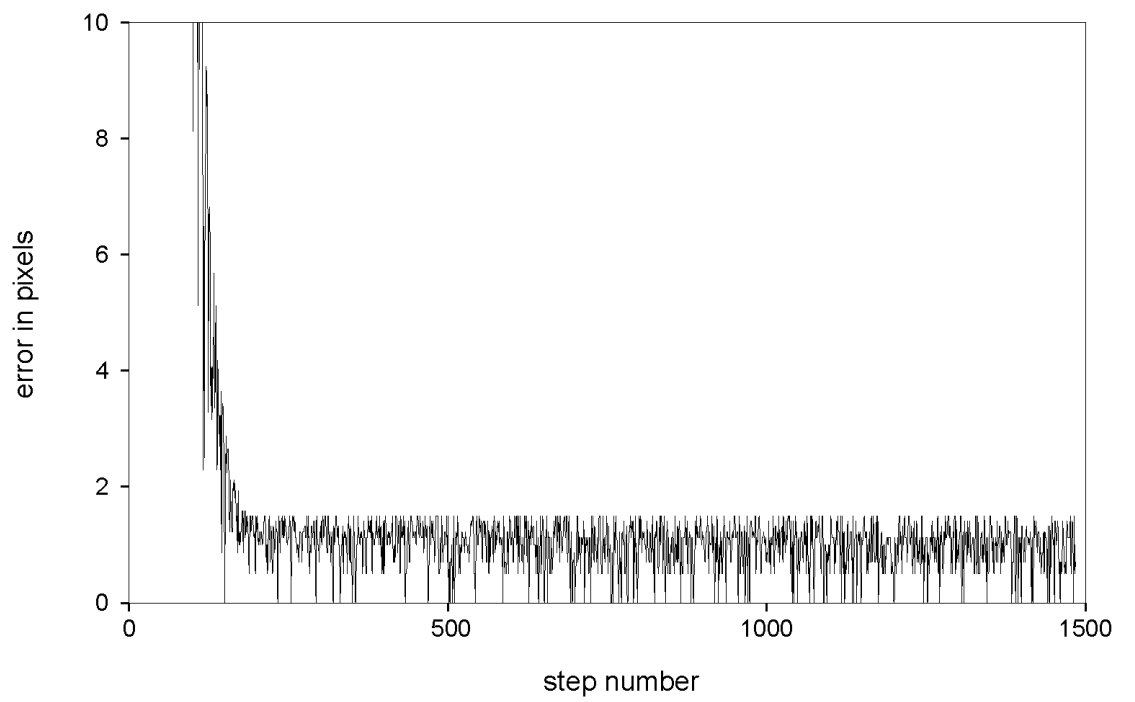


Figure 7: Final error between position and target, in pixels, vs step number. One pixel corresponds to  $\sim 3$  mm.

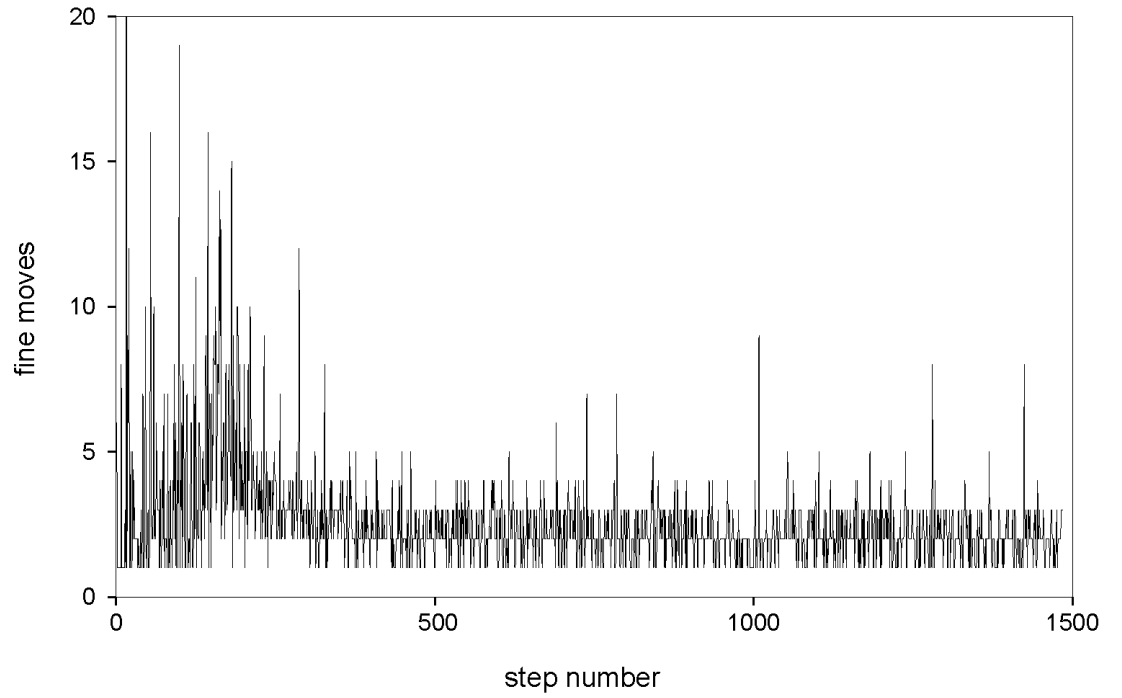


Figure 8: Number of fine movements required to reach tolerance defined in eq. (13), vs step number

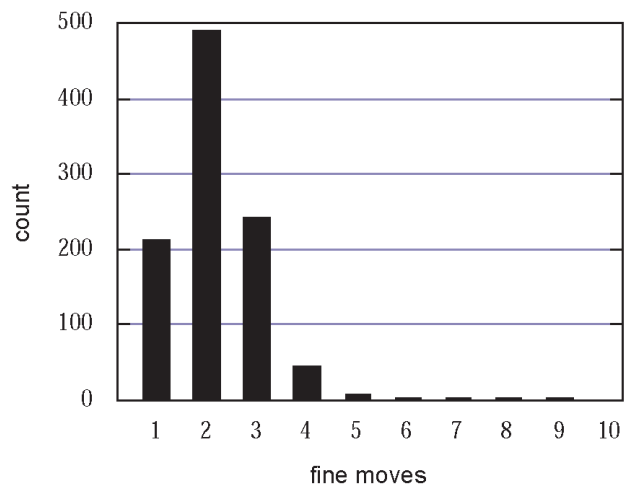


Figure 9: Histogram of number of fine movements required to reach the tolerance defined in eq. (13) for steps 485 to 1484.

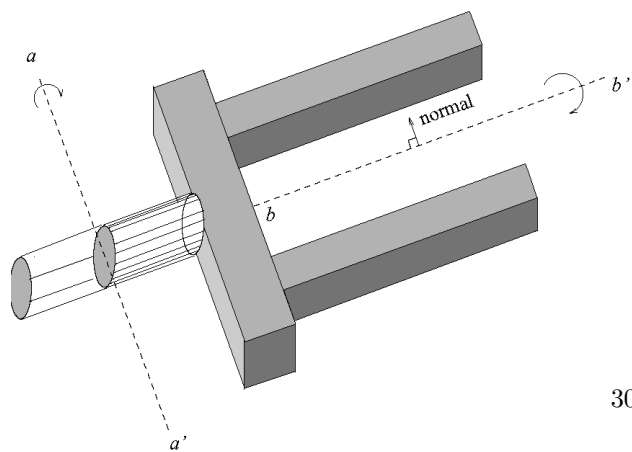


Figure 10: A sketch of the gripper with two types of associated motions

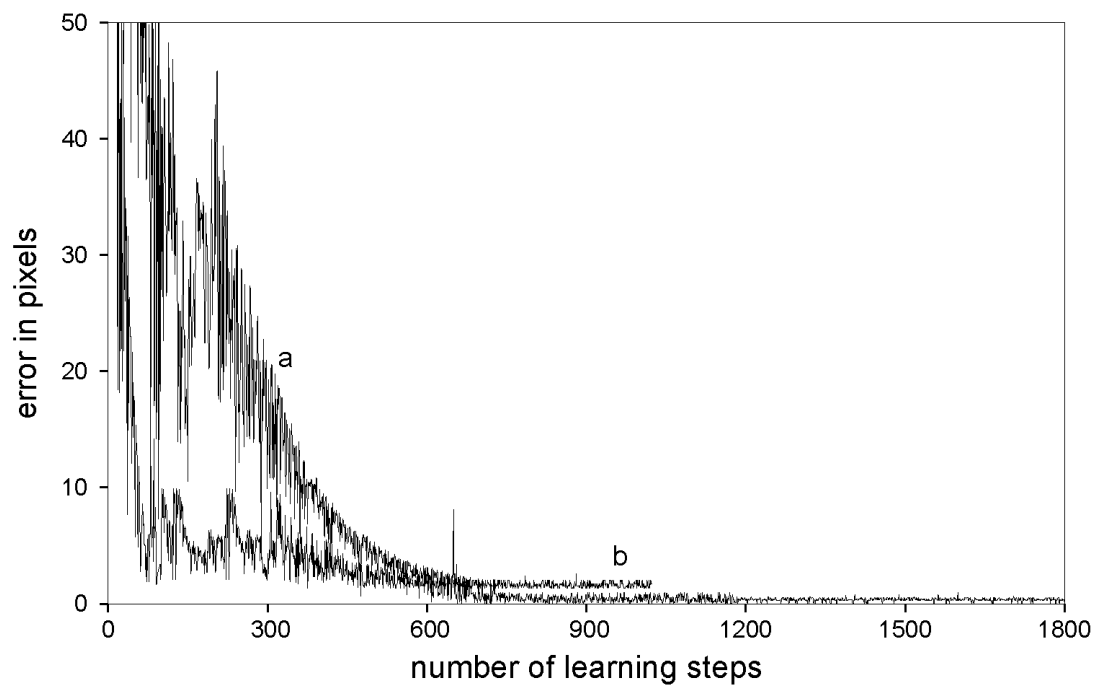


Figure 11: Positioning and orientation error a: positioning error; b: orientation error