Robot hand-eye coordination using neural networks

TR CS–93–10

Patrick van der Smagt Frans Groen Ben Kröse
University of Amsterdam
Department of Computer Systems
Kruislaan 403
1098 SJ Amsterdam
email <smagt@fwi.uva.nl>

October 25, 1993

This paper focuses on static hand-eye coordination. The key issue that will be addressed is the construction of a controller that eliminates the need for calibration. Instead, the system should be self-learning and must be able to adapt itself to changes in the environment.

In this application, only positional information in the system will be used; hence the above reference 'static.' Three coordinate domains are used to describe the system: the *Cartesian world-domain* with elements $\vec{x} = (x, y, z)$, to describe the position of the target object and the robot end-effector in world coordinates; the *vision domain* with elements $\vec{\xi}$, to describe the observation of the target object relative to the camera position; and the *robot domain* with elements $\vec{\theta}$ known as the **joint values** of the robot, to describe the position of the robot end-effector relative to the robot base.

The task that is set out to be solved is the following. A robot manipulator has to be positioned directly above a pre-specified target, such that it can be grasped. The target is specified in terms of visual parameters. Only the (x, y, z) position of the end-effector relative to the target is taken into account; this suffices for many pick-and-place problems encountered in industry. (In a number of cases, also the rotation of the hand is of importance, but this rotation can be executed separate from the 3D positioning problem.) Thus the remaining problem is 3 degrees-of-freedom (DoF).

Of the robot no explicit model is needed, but only some basic assumptions (explained below) are incorporated. It is known that the robot is controlled by specifying changes of its joint values $\vec{\vartheta}$, and that the 'current' joint values $\vec{\theta}$ can be measured. The $\vec{\vartheta}$ are delta values of $\vec{\theta}$; i.e., $\vartheta_i \equiv \Delta\theta_i$. Secondly, from the visual system it is only known that it gives positional information $\vec{\xi}$ of the observed object.

Given the visual data $\vec{\xi}$ and the robot positional data $\vec{\theta}$, a model-free adaptive controller learns to generate robot commands $\vec{\vartheta}$ which position the end-effector directly above the target object.

We are interested in the relation between the visual features (i.e., positions of point features) $\vec{\xi}$ on the one hand, and joint positions $\vec{\theta}$ on the other. Traditional model-based hand-eye coordination tends to consider this as two separate problems: first, a translation from visual to Cartesian domain, followed by a translation from Cartesian to joint domain (called **inverse kinematics**). This two-stage approach is chosen since it simplifies calibration, and allows separate camera and robot models.

In the model-free and calibration-free approach proposed here, however, a sensor reading $(\vec{\xi}, \vec{\theta})$ can be directly mapped onto a robot motion $\vec{\vartheta}$. The system is thus set up that the relation

$$(\vec{\xi}, \vec{\theta}) \longrightarrow \vec{\vartheta} \tag{1}$$

is many-to-one, i.e., given a sensor reading, a robot motion can be uniquely determined. This simplification means that there is always only one posture (arm configuration) to reach a specific

position; for instance, there is no elbow up–elbow down ambiguity. This situation is realised by choosing a preferred situation (c.q., the elbow up configuration) and never teaching the conflicting (elbow down configuration) to the adaptive controller. Note that it is not guaranteed that the elbow down configuration will never occur; when the controller parameters are sufficiently disturbed, this case is, in theory, not excluded.

The visual techniques required to accomplish the set task are discussed in section 1. Section 2 describes control of the robot to this end. Then, section 3 shows how the visual and motion data is combined in the system which controls the robot to reach the target.

Section 4 describes the method to obtain instances of the many-to-one mapping from sensor reading to robot motion using a self-supervised technique. The maximum attainable precision that can be obtained with the camera/robot system is discussed in section 5. Finally, details of the neural controller are described in 6, and results are given in section 7.

1 Robot Perception

In hand-eye coordination, the motion plan is based on a visual observation. For the system to accomplish the specified task, it must know the position of the object relative to the robot, in some coordinate system. There are two basic visual setups to reach this goal:

1. **External 'world-based' camera:** both the robot and target object are observed by cameras situated at fixed and unchanging positions. There must be a one-to-one or many-to-one relationship between the observations (a set of visual point features of the object) $\{\vec{\xi}_i\}_o$ and position $\vec{x}_o$ of target object; otherwise, when the target position is not uniquely known, the required robot move to reach the target cannot be determined from the observation. Also, the visual observation (a set of visual point features of the robot) $\{\vec{\xi}_i\}_r$ of the robot end-effector must uniquely determine its world coordinates $\vec{x}_r$, to ensure that the relation between the vision domain and the robot domain is learnable.

2. **Internal 'robot-based' camera:** the target object is observed by cameras which move together with the robot's end-effector. The relation visual object position–Cartesian object position can now be many-to-many. However, this time the measurements $(\vec{\theta}, \vec{\xi})$ must uniquely define the world coordinates $\vec{x}_o$ of the object in the case of static positioning as defined in the introduction of this paper.

These two setups are basically different in the following way. In the case of world-based vision, the target object, the robot end-effector, and the (positional) relation between the two must be determined. In robot-based vision the target object is observed in a camera coordinate frame relative to the robot position $\vec{\theta}$. Therefore it is not necessary that the position of the robot end-effector be observed, and hence robot-based vision is simpler and more robust.

Robot-based vision has another advantage over world-based. The positional precision that can be extracted from a discretised camera image is inversely proportional to the distance between the camera and the observed scene. World-based cameras, which are not moving with the gripper and yet have to see the whole work space, have to be placed rather far away, typically in the order of 2m, from the robot base. Thus their precision will be limited, typically 0.5–1cm for the application in this paper. When a robot-based camera is used, however, the precision increases when the target object is approached. The limited resolution of the camera is no limiting factor.

The perspective transformation that maps 3D points on a plane (c.q., the image plane) is a many-to-one relationship and is thus not invertible. A single image obtained from one camera does not provide the required depth information to reconstruct the positions of the observed components in the 3D world. Since the controller used in this paper needs such information, an extension is required. Two solutions are considered.

1. **Model-based monocular:** in this case, *a priori* knowledge of the observed object is assumed. For example, assume that two visual point features A and B are observed with the

camera. When the camera is at a predetermined (i.e., constant but not necessarily known) distance from, and has a predetermined (i.e., constant) angle with the line A–B, the positions of points A and B relative to each other is measured. Then there is a one-to-one relationship between the observed relative distance Δ_{AB} between A and B, and the depth d:

$$d = k/\Delta_{AB} \tag{2}$$

where k is a constant. This is the principle of obtaining depth from a monocular image with *a priori* knowledge. For more detailed information about robust model-based approaches, see (Kanade, 1981).

2. **Correspondence-based binocular:** A stereoscopic system solves the problem, too (Ballard & Brown, 1982; Fu, Gonzalez, & Lee, 1987). When two cameras, whose image planes are situated at known (relative) positions, observe the 3D scene, the corresponding point features from both images can be used to reconstruct the 3D image. (As shown in, e.g., (Ritter, Martinetz, & Schulten, 1989), the relative image plane positions need not be calibrated but can be incorporated in the learning mechanism.)

   When the visual scene becomes increasingly complex, or the number of degrees of freedom for positioning the robot manipulator increases (e.g., not only the position but also the orientation of the hand is of importance), the complexity of stereo vision will increase considerably. Especially for complex visual scenes, the **correspondence** or **matching** problem (Marr, 1982; Ballard & Brown, 1982) becomes significant: which point features in the left image correspond with which point features in the right image?

So, in order to circumvent the correspondence problem, model-based vision is preferred. However, apart from the limited precision, there is another reason that this does not combine well with an external camera. It is very cumbersome to detect two point features on a moving body (the robot manipulator) which maintain the same orientation with the image plane.

In conclusion, a robot-based camera, viz. a camera mounted in the gripper of the robot, combined with model-based vision is the preferred setup. This setup gives the following advantages:

- no correspondence problem;
- increasing visual precision when the target is approached;
- simple image processing.

The only remaining disadvantage is that *a priori* knowledge of the object is required for static positioning.

Figure 1 shows the placement of the camera and optical system in the centre of the robot's gripper.

1.1 Properties of the visual system

Given the placement of the camera, and the data that is obtained from it, the visual processing system can be discussed in more detail. The system must accomplish the following tasks (Fu et al., 1987):

1. **image acquisition**: the image, projected on the camera's image plane, must be transferred to the memory of the image processing hardware. This renders a discretised image $\mathcal{I}$;
2. **image segmentation**: from the discrete image $\mathcal{I}$ it is determined which parts represent components, and which represent background. This is typically done using one of the two basic principles: continuity and similarity;
3. **image description**: for the purpose of recognition and for subsequent use in the control algorithm, for each component features $\vec{\xi}$ describing the properties of the component as well as its position are determined;

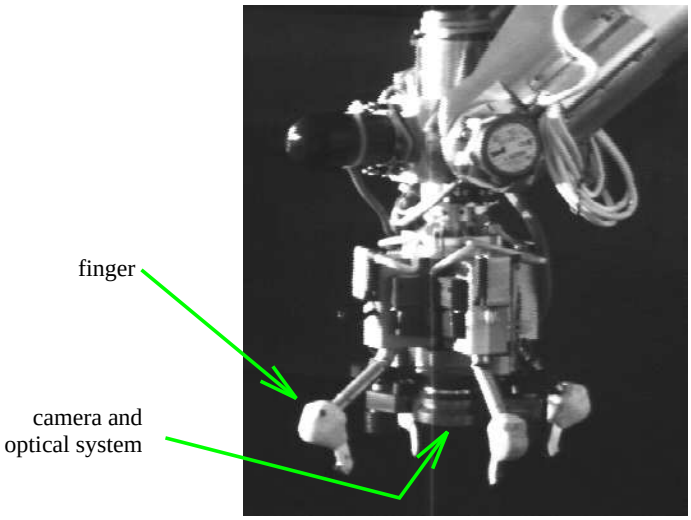


Figure 1: Camera placed in the manipulator of the robot.

4. **component recognition**: the identified components are labeled as target object, robot hand, background, ... For the purpose of this paper, only the target object and the background need to be identified.

It is furthermore required that the visual processing forms not the bottleneck in the control flow. The time delay presented by the camera system has a lower bound of 40ms, which is the time needed for image acquisition. During acquisition of $\mathcal{I}[i+1]$, the image processing system has time to extract point features from image $\mathcal{I}[i]$. By keeping the visual scene sufficiently simple, the required point features can be extracted from the approximately 512×512 pixels of 8 bits width within 40ms by using a DataCube MaxVideo–20 image processing board. It is therefore chosen that the visual scene consists of a single white blob against a dark background. From this blob, the (x, y) position (ξ_x, ξ_y) on the image as well as its area ξ_A are extracted. Details of this operation are described in (Bartholomeus, 1993). These three point features portray all information for the 3D localisation of the object; however, choice of these point features is not essential for the control algorithm. The simple visual setup is chosen to get a good performance in the preprocessing and suffices for the current research purposes; for the neural controller, it is only essential that the visual features are complete, i.e., contain all necessary information.

2 Robot Motion

The neural control algorithm, which accepts a camera observation $\vec{\xi}$ and a robot state $\vec{\theta}$ as input, generates a robot move $\vec{\vartheta}$. The transformation between the coordinate frame of the $\vec{\xi}$ coincides and the coordinate frame of the gripper is constant, i.e., the visual observations are done with respect to the 'current' robot position. Therefore, in order to get complete information about the position of the target, the $\vec{\alpha}$ itself does not suffice but needs to be complemented with $\vec{\theta}$ (cf. eq. (1)). Note that, due to the position of the camera in the robot's end-effector, and the structure of the robot (see below), $\vec{\theta}_1$ is not needed as controller input.

In this section, the definition of $\vec{\vartheta}$ will be examined. A distinction between the *static* and the *dynamic* description is made. With a **static description**, the position of the robot is considered in terms of its joint values. That it, for a given *stiff* robot, there exists a one-to-one correspondence between the robot's joint values and the position of its end-effector relative to its base. This correspondence is known as the robot's **kinematics**.

Secondly, the **dynamics** are concerned with the forces (torques) necessary to move the robot. A robot must be moved by applying torques to its joints. The motion that is induced is generally

described in terms of the variables $\vec{\vartheta}$, $\dot{\vec{\vartheta}}$, and $\ddot{\vec{\vartheta}}$, and parameter matrices describing the inertia, the Coriolis coefficients, centrifugal coefficients, friction, and gravity. The relationship between the path followed in joint space (i.e., in terms of $\vec{\vartheta}$, $\dot{\vec{\vartheta}}$, and $\ddot{\vec{\vartheta}}$) and the torques applied to the joints is known as the robot's **dynamics**. For a stiff robot, this relationship can usually be determined within acceptable accuracy, such that a controller can be made which calculates the desired torques to effectuate a path in joint space.

In this paper, only knowledge of the position of the arm will be used in the controllers. It is therefore chosen that the dynamics be neglected. This means that it is assumed possible to control the robot arm by specifying paths in joint angles rather than torques; the underlying robot dynamics controller, currently not of interest, will actually perform the calculations necessary to move the arm. Section 5.3 will discuss the influence of the dynamic controller used in the experiments.

2.1 OSCAR kinematics

A flexible way of describing robot kinematics is by means of the Denavit-Hartenberg notation. With each joint i of the robot, four parameters θ_i, d_i, a_i, and α_i are associated (Craig, 1986; Fu et al., 1987). For a rotational joint, θ_i is variable; for a translational joint, d_i is the variable. All the other parameters are constant for that particular joint. Figure 2 shows the definitions of the parameters.

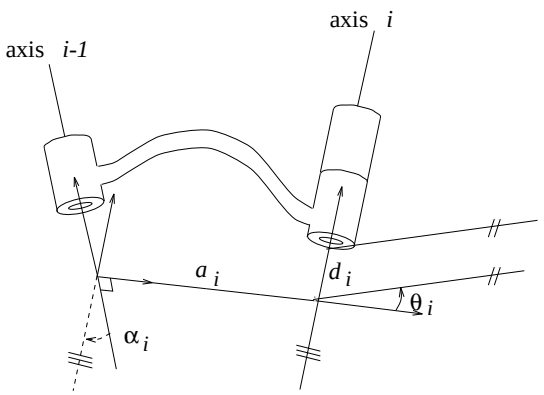


Figure 2: Parameters θ_i, α_i, d_i, and a_i for joint i.
In this case, the joints are rotational, such that the corresponding θ's are variable and the other parameters are constant.

When with each link i a coordinate frame is associated, the transformation from the coordinate frame attached to link $i-1$ to the coordinate frame attached to link i is given by the homogeneous matrix

$$ {}^{i-1}A_i = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3} $$

The position and orientation of link i relative to the base frame is then given by the matrix

$$ {}^0A_i \equiv {}^0A_1 {}^1A_2 \cdots {}^{i-1}A_i. \tag{4} $$

The last column of this matrix gives the *position* of the associated coordinate frame; when this frame is attached to the end-effector, the position of the robot's hand can be found.

Equation (4) describes the **forward kinematics** of the robot. The inverse problem, of finding a set of joint values to attain a specified end-effector position and orientation, is called the **inverse kinematics**. This problem, although more difficult to solve, is computable if the target state is

i	θ_i	d_i	a_i	α_i
1	*	46cm	0cm	$-90°$
2	*	0cm	51cm	$0°$
3	*	0cm	0cm	$90°$
4	*	50cm	0cm	$-90°$
5	*	0cm	0cm	$90°$
6	*	21cm	0cm	$0°$

Table 1: Denavit-Hartenberg parameters for the OSCAR–6 robot.
An '*' indicates that the corresponding parameter is a variable.

uniquely described in terms of position and orientation of the coordinate frames. Often, however, only the position and orientation of the end-effector is of important, in which case multiple solutions (such as elbow-up and elbow-down) exist. This is solved by simply *choosing* a preferred solution; such a choice is often given by the underlying path planner, e.g., by taking the shortest path from the current to the target position.

The robot used in the experiments is a 6 DoF anthropomorphic robot called OSCAR–6. Figure 3 gives a schematic representation of this robot, showing the coordinate frames and the definitions of the joint angles. The corresponding Denavit-Hartenberg parameters are given in table 1.

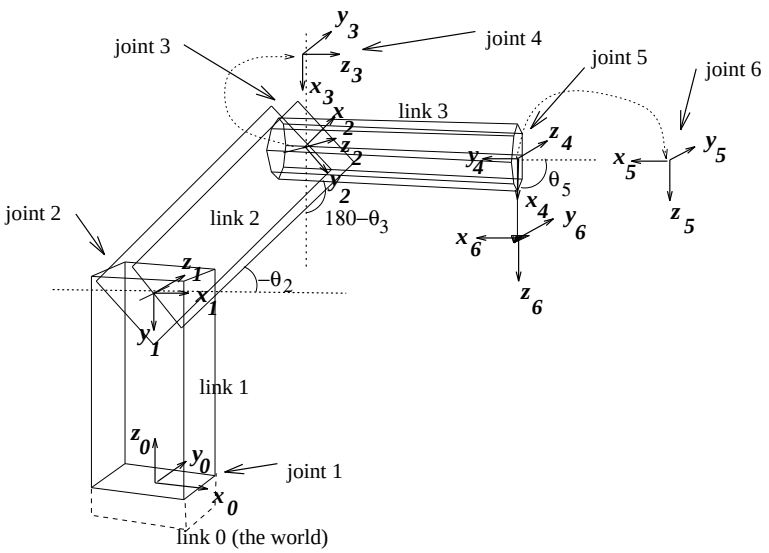


Figure 3: Line drawing representation of the OSCAR–6 robot used in the experiments.

As defined in the introduction, the control system is restricted to three DoF. Therefore, only three of OSCAR's six joints are controlled by the adaptive controller. Specifically, joints 4 and 6 are always kept at a constant value; joint 5 is thus adjusted that the camera, fixed in the robot's hand, is always looking down.

3 System Description

Having qualitative models of the sensors and the robot, a controller must be designed to solve the task set out in the introduction.

The design of a controller depends on the knowledge that is available from the process that is to be controlled. Three stages of control are distinguished (Bellman & Kalaba, 1959; Narendra & Annaswamy, 1989):

1. when the controller has complete information about the behaviour of the inputs of the process, and this process is fully specified, the process is called a **deterministic control processes**;

2. when unknown variables are present as random variables with known distribution functions, the process is called a **stochastic control process**;

3. finally, when even that information is not available, but the controller must learn to improve its performance by observing the behaviour of the controlled process, this process is referred to as an **adaptive control process**.

Clearly, within the scope of the task set out in the introduction, the process that is controlled is in the third of the above categories.

An adaptive controller has the problem of the following duality (Feldbaum, 1965; Narendra & Annaswamy, 1989): first, it must *identify* the process that is to be controlled and find its parameters, and secondly, it must find which actions are required for *control* of the process. Two solutions exist (Narendra & Annaswamy, 1989; Narendra & Parthasarathy, 1990; Åström & Wittenmark, 1989): **direct adaptive control** and **indirect adaptive control**. In indirect control (figure 4b), the parameters of the plant are estimated on-line, and these estimates are used to update the parameters of the controller. In direct control, however, plant parameters are not estimated but the control parameters are directly updated to improve the behaviour of the system (figure 4a).

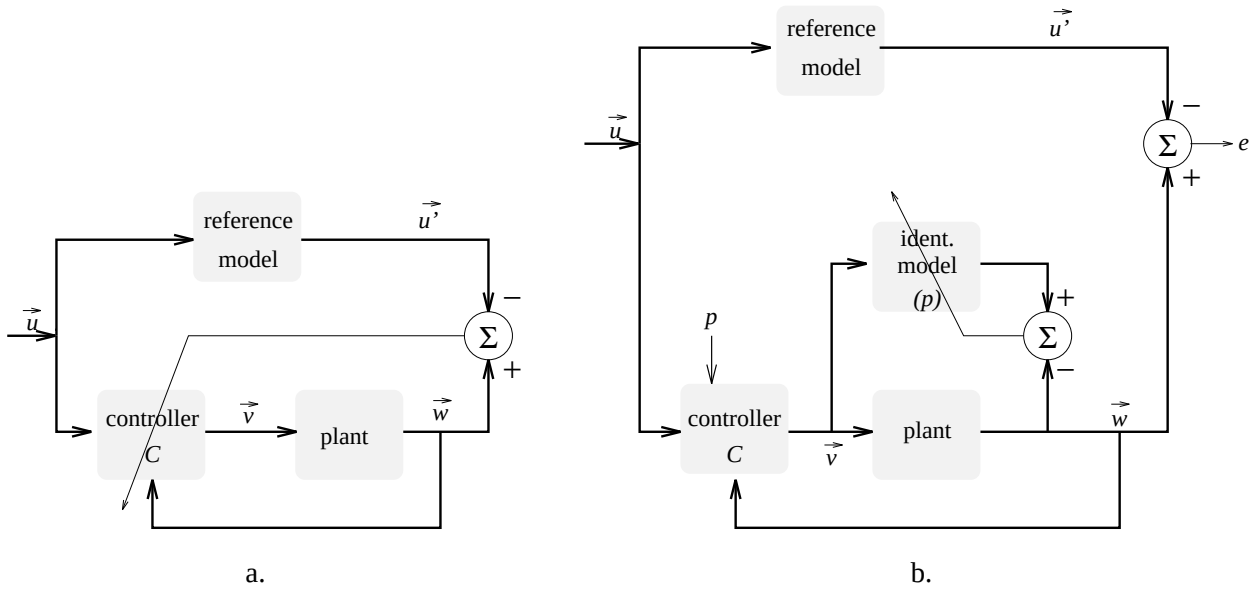


Figure 4: a. Direct adaptive control. b. Indirect adaptive control.

The direct control method in figure 4a work as follows. The plant is controlled by a signal $\vec{v}$, and outputs a signal $\vec{w}$. The reference signal is a signal $\vec{u}$ which is input to the controller; the task for the controller C is to generate a signal $\vec{v} = C(\vec{u})$ which minimises $\|\vec{u}' - \vec{w}\|$. The $\vec{u}'$ is derived from $\vec{u}$ by means of the **reference model**, as shown in the figure. Differently put, the reference model translates the $\vec{u}$ to the domain of $\vec{w}$, such that the two can be combined in a single error signal. This error signal is subsequently used in the update of the controller.

The direct control method is normally not used for the following reason. The error signal $\|\vec{u}' - \vec{w}\|$ carries information on how the output of the *plant* must change, and not the output of the *controller*. In order to determine the latter, the error has to be back-propagated through the plant, thus requiring an inverse model of the plant. Obviously, since this model is not available this approach is not viable.

A solution is given by using the **indirect control** method. An extra model called the **identification model** is introduced in the system, which learns to copy the behaviour of the plant, i.e., it is a forward model of the plant. This model will be some parametric model of which the resultant parameters p describe the behaviour of the plant. These parameters are then used in the update of the controller. The term 'indirect control' is applied since the controller does not directly minimise the output error e (figure 4b), but rather the error which is fed back into the identification model.

Choosing a controller. We consider the first, direct control, approach preferable since only one set of parameters needs adaptation, and only one model is required. Also, by directly expressing the reference signal $\vec{u}$ (in this case, $\vec{\alpha}$) in terms of the output of the plant, the reference model can be eliminated. This can be done because the task of the robot is expressed in terms of visual parameters, and the plant's output consists of the same visual parameters. It will then result in an error signal $\|\vec{u} - \vec{w}\|$ expressed in the *input* domain of the controller instead of its output. It will be shown below that this still leads to a useful error signal such that the controller can be updated.

For this specific case, the output of the plant is a visual state $\vec{\xi}$ together with a robot state $\vec{\theta}$. Both of these are used in the feedback to the controller, but only $\vec{\xi}$ is combined with the desired plant output $\vec{\alpha}$ into an error signal. The resulting controller is shown in figure 5.

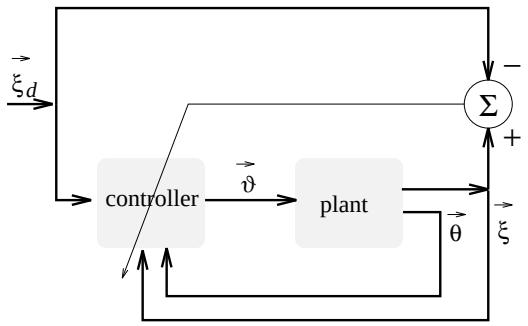


Figure 5: The controller used in the experiments.

3.1 The experimental setup

The above ideas need further specification. The proposed experimental setup is shown in figure 6.

The system will be described in what will be called discrete **steps** i, $i+1$, ... These steps need not be equidistant in time, i.e., there is a non-linear correspondence between the time scale and the steps i. Furthermore, the terms 'move' and 'goal' are introduced here. A **planned move** or **move** for short is a single $\vec{\vartheta}$ generated by the controller which causes the robot to move. A move may consist of several steps. A move need not be finished but may be interrupted by another move. Finally, a **goal** is a sequence of moves which bring the robot from its starting position to the position where the target is considered 'reached.'

The task that has to be performed can be described in terms of the visual observation $\vec{\xi}$ with respect to time. Thus, the definition of grasping is reaching the state where

$$\forall i > n_\tau : \vec{\xi}[i] = \vec{\alpha} \tag{5}$$

where n_τ is an algorithm-dependent moment in time. The $\vec{\alpha}$ is a setpoint for $\vec{\xi}$. It consists of the desired point feature values of the object observation when the object is considered 'reached.' In particular, for the presented controller the setpoint corresponds with the extracted visual point features ξ_x, ξ_y, and ξ_A when the blob is at a predetermined size located in the visual centre of the image.

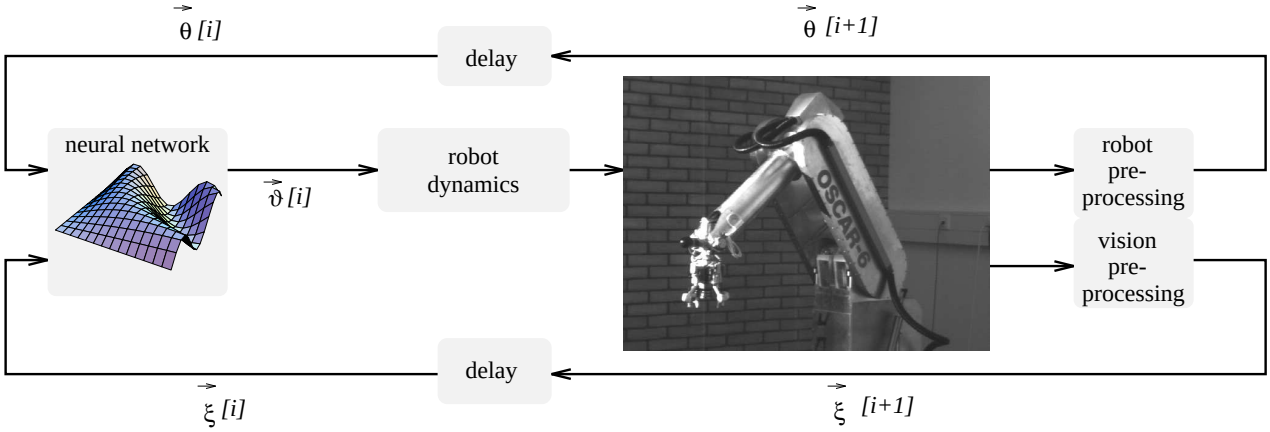


Figure 6: The experimental setup.

The robot–camera system (henceforth called **animat**) is represented by $\mathcal{R}_n(\cdot)$. From a starting joint position $\vec{\theta}[i]$, given a move $\vec{\vartheta}[i]$ the animat will move in one step to a joint position $\vec{\theta}[i+1]$ while the object observation moves from $\vec{\xi}[i]$ to $\vec{\xi}[i+1]$:

$$\mathcal{R}_1\left(\vec{\theta}[i], \vec{\vartheta}[i], \vec{\xi}[i]\right) = \left(\vec{\theta}[i+1], \vec{\xi}[i+1]\right). \tag{6}$$

The subscript '1' indicates one time step. The definition of $\mathcal{R}$ can be extended to allow more steps to be made per move, resulting in a recursive definition of $\mathcal{R}_n$:

$$\mathcal{R}_{n+1}\left(\vec{\theta}[i], \vec{\vartheta}[i], \vec{\xi}[i]\right) = \mathcal{R}_n\left(\vec{\theta}[i+1], \vec{\vartheta}[i] - (\vec{\theta}[i+1] - \vec{\theta}[i]), \vec{\xi}[i+1]\right) \tag{7}$$

which follows from the definition of $\mathcal{R}_1$. Unless otherwise indicated, the subscript '1' may be left out.

An example path followed by the robot in one goal is shown in figure 7. Clearly, the $\vec{\vartheta}$'s

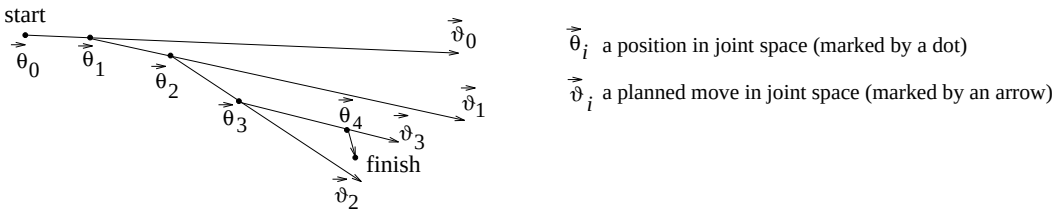


Figure 7: 2D motion plan of the robot arm.
A move $\vec{\vartheta}$ is, in general, not completed. While the robot is moving towards the new setpoint, a new setpoint is received, and the motion plan is updated.

which are received by the animat will not be reached by the time a new move $\vec{\vartheta}[i+1]$ is received by the animat. The motion plan (i.e., set of via points through which the robot moves) will be recomputed such that the robot moves from the current position $\vec{\theta}[i+1]$ towards the target position $\vec{\theta}[i+1] + \vec{\vartheta}[i+1]$.

At some step i, both $\vec{\xi}[i]$ and $\vec{\theta}[i]$ are known. To prevent the necessity of an interpolation algorithm, these measurements are thus clocked that, indeed, they are obtained at the same time (within the resolution of the system). These measurements are then fed into the neural controller, which then generates a robot motion $\vec{\vartheta}[i]$. The neural controller is represented by $\mathcal{N}$, which generates a joint rotation $\vec{\vartheta}[i]$ when given joint position $\vec{\theta}[i]$ and visual position $\vec{\xi}[i]$:

$$\mathcal{N}\left(\vec{\theta}[i], \vec{\xi}[i]\right) = \vec{\vartheta}[i]. \tag{8}$$

Using these definitions, both $\mathcal{R}$ and $\mathcal{N}$ are bijections.

The goal of teaching the neural network is to obtain an ideal controller $\mathcal{N}_0$. This network generates a joint rotation which makes the visual position $\vec{\xi}[n_\tau]$ after the robot move equal to $\vec{\alpha}$:

$$\mathcal{R}_\infty\left(\vec{\theta}[i], \mathcal{N}_0\left(\vec{\theta}[i], \vec{\xi}[i]\right), \vec{\xi}[i]\right) = \left(\vec{\theta}[n_\tau], \vec{\xi}[n_\tau] = \vec{\alpha}\right). \tag{9}$$

The subscript ∞ indicates the equality for a completed robot move.

As explained above, the $\mathcal{N}$ does not depend on a specific model of the $\mathcal{R}$, but is a general parametric model which is trained with learning samples obtained during operation. The method of generating the learning samples defines the 'reference model' that needs to be used.

4 Generating learning samples

The neural network $\mathcal{N}(\cdot)$ is trained with learning samples $(\vec{\theta}, \vec{\xi}; \vec{\vartheta})$. Unfortunately, it is not possible to generate a $\vec{\vartheta}$ from a given $(\vec{\theta}, \vec{\xi})$, since that would require knowledge of the inverse robot $\mathcal{R}^{-1}(\cdot)$. This is an acknowledged control theory problem, which instigates researchers not to use direct control (Narendra & Parthasarathy, 1990). To alleviate this problem, an alternative is presented in this paper: given a $\vec{\vartheta}$, find the corresponding $(\vec{\theta}, \vec{\xi})$. This is equivalent to finding a sample which corresponds with the ideal network in eq. (9): $\mathcal{R}_\infty(\vec{\theta}[i], \vec{\vartheta}[i], \vec{\xi}[i]) \rightarrow (\vec{\theta}[i+1], \vec{\alpha})$.

4.1 The input adjustment method

However, normally there are only measurements $\mathcal{R}(\vec{\theta}[i], \vec{\vartheta}[i], \vec{\xi}[i]) \rightarrow (\vec{\theta}[i+1], \vec{\xi}[i+1])$. A first onset is similar to a method presented in (Psaltis, Sideris, & Yamamura, 1988) which works as follows. Suppose a measurement $(\vec{\theta}[i], \vec{\xi}[i], \vec{\xi}[i+1]; \vec{\vartheta}[i])$ is available. By imaginarily rotating and translating the camera over T such that the $\vec{\xi}[i+1]$ is translated to $\vec{\alpha}$, we have $T\vec{\xi}[i+1] = \vec{\alpha}$. The same T is used in translating $\vec{\xi}[i]$. However, T is expressed in the image frame of $\vec{\xi}[i+1]$, so before it can be applied to $\vec{\xi}[i]$ the latter must be expressed in the same image frame. This is done through the rotation matrix ${}^{i}R_{i+1}$ which expresses the rotation of the image from $\vec{\xi}[i]$ to $\vec{\xi}[i+1]$. Thus a learning sample

$$\left(\vec{\theta}[i], \vec{\xi}[i] - {}^{i}R_{i+1}\vec{\xi}[i+1]; \vec{\vartheta}[i]\right) \tag{10}$$

is obtained. This method, called the **input adjustment method** and introduced in (Kröse, van der Korst, & Groen, 1990), is illustrated in figure 8. Since the rotation is only about the z-axis, the rotation matrix can be calculated from θ_1, the rotation about the robot base. Given an adequate shape of the object (e.g., a non-isosceles triangle), the rotation can also be visually derived.

5 Attainable precision with the camera/robot system

Since the controller is an interface between real-world systems, errors in measurements are bound to occur. Some of these errors will have more severe effects than others. It is therefore necessary to re-examine the input-adjustment method and evaluate the effect of measurement errors on the construction of learning samples.

Then, before determining which precision the controller has to attain, it is necessary to determine at which precision the animat can be controlled. There are two uncertainties to take into account: the camera precision and the robot precision. The robot inaccuracies are due to imperfections in the dynamic controller, and noise in the joint position measurements. The visual measurements suffer from noise, most notably in the area measurements.

In this section, these quantities are investigated,

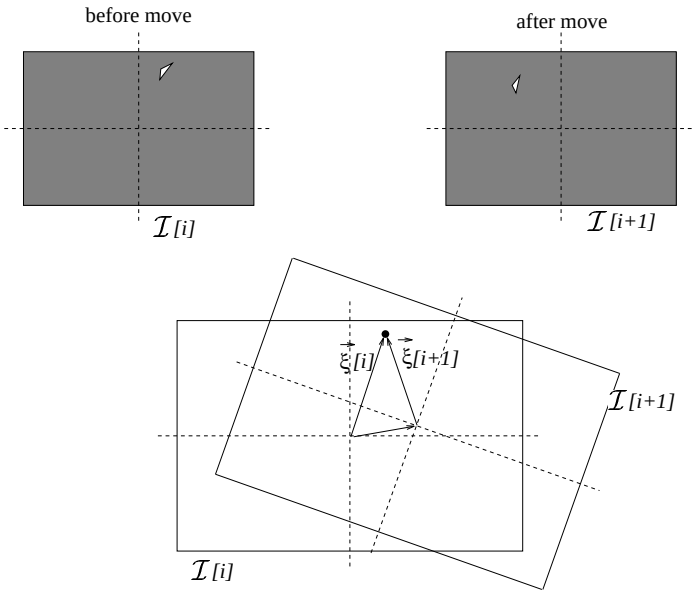


Figure 8: The position of the target in the coordinate frames before and after the movement of the arm.

5.1 Error in position

The input adjustment method works by translating the move $\vec{\xi}[i] \rightarrow {}^{i}R_{i+1}\vec{\xi}[i+1]$ to a move $\vec{\xi}[i] - {}^{i}R_{i+1}\vec{\xi}[i+1] \rightarrow {}^{i}R_{i+1}\vec{\xi}[i+1] - {}^{i}R_{i+1}\vec{\xi}[i+1] = 0$. This will generate a correct learning sample if the projection of the Cartesian world coordinates onto the CCD ($\vec{\xi}$) is linear in the translation over ${}^{i}R_{i+1}\vec{\xi}[i+1]$. However, due to the fact that a wide-angle lens is used, a translation from point A to point B in Cartesian domain, resulting in a move $\vec{x}_B - \vec{x}_A$, will give a translation in visual domain $\vec{\xi}_B - \vec{\xi}_A$ which depends on where in the camera image this translation finds place.

A camera model. How serious is this deformation? To obtain an estimate, a model due to (Lagerberg, van Albada, & Visser, 1993), which is based on measurements with the OSCAR camera, is used. According to this model, the following deformations have to be taken into account:

1. A point (x_c, y_c, z_c) in the camera coordinate system (which coincides with the robot end-effector coordinate system) is projected on a point (u, v, w) in the image plane coordinate system as
$$\begin{pmatrix} 509.5 & -2.47 & 0.0 \\ 0.0 & -619.5 & 0.0 \\ -0.0735 & -0.0484 & 1.0 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}.$$
This means that there is an angle of 5.0° between the normal to the image plane and the z axis of the camera coordinate system.

2. The optical centre of the camera is not the centre of the CCD. These centres are located at a distance $(\Delta x = 46, \Delta y = 30)$ pixels from each other.

3. Deformation in the x and y direction due to the wide-angle lens. The x, y coordinate which would have been obtained with a pinhole camera is projected on the deformed x_v, y_v on the CCD as
$$x_v = x - \frac{k_1 r^2 (x - \Delta x) + k_2 r^4 (x - \Delta x)}{1 + k_1 r^2 + k_2 r^4} \tag{11a}$$

$$y_v = y - \frac{k_1 r^2 (y - \Delta y) + k_2 r^4 (y - \Delta y)}{1 + k_1 r^2 + k_2 r^4} \tag{11b}$$

$$r^2 = (x - \Delta x)^2 + (y - \Delta y)^2 \tag{11c}$$

where $k_1 = 6.24 \cdot 10^{-7}$, $k_2 = -4.97 \cdot 10^{-13}$.

When the input adjustment method is used, errors 1 and 3 will introduce systematic errors in the learning samples. The effect of error 1, which is a cosine relationship resulting in maximally one pixel inaccuracy, is negligible compared to error 3. The effect of error 3 can be measured by calculating the Euclidian pixel distance between the ideal and deformed pixel $\sqrt{(x_v - x)^2 + (y_v - y)^2}$. As is shown in figure 9, a measurement can be as much as 40 pixels or 8% off. For small moves, this error will occur in both the start and end position, and will therefore not be disastrous. Only for moves across the whole image, an 8% error will result.

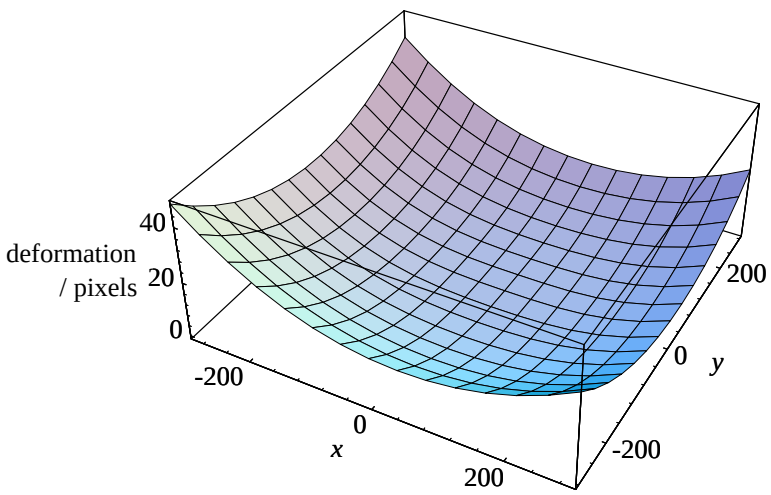


Figure 9: Deformation in the camera image.
The deformation is calculated with the model described in (11) and is given by the Euclidian distance between (x, y) and (x_v, y_v).

5.1.1 Solutions to the error problem

Two solutions to solve the above problems are introduced. First, it is possible to calculate the *goodness* of a learning sample. Applying the input adjustment to a move introduces a systematic error, dependent on how far the animat is from the target state after making this move. A sample is perfect if the corresponding move has made the robot to reach the target state. The 'error' in every sample can be weighed by the distance $\|\vec{\alpha} - \vec{\xi}\|$ where $\vec{\xi}$ is the visual state after the corresponding move. These samples, which are used in finding the optimal parameters of the neural controller, are then collected in such a way that samples with a smaller error replace samples with a larger error. In the implementation, a binary approach is taken: 'perfect' learning samples, those that bring the animat 'close enough' to the target state always replace 'imperfect' samples. On all

the samples, the input adjustment method is applied. With figure 9, the error introduced in the 'perfect' learning samples is very small, since the camera deformation around the camera centre is negligible.

Such 'perfect' samples will become available as soon as the neural network becomes successful in controlling the robot. When these samples appear, the set of learning samples will be updated by having perfect learning samples replacing imperfect ones.

The second solution involves a change in the neural controller. An extra set of inputs is introduced such that $\mathcal{N}'$:

$$\mathcal{N}'\left(\vec{\theta}[i], \vec{\xi}[i], \vec{\xi}[i+1]\right) = \vec{\vartheta}[i] \tag{12}$$

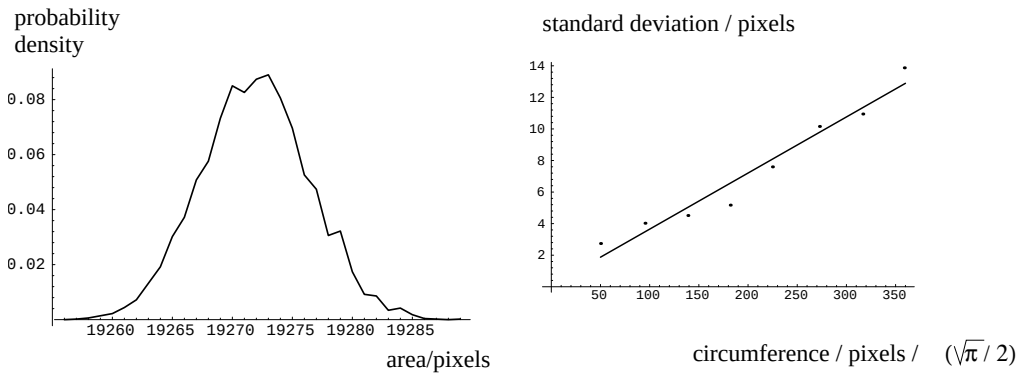
where $\vec{\xi}[i]$ is the visual state before the move, and $\vec{\xi}[i+1]$ after the move. When the network is used to generate a new $\vec{\vartheta}[i+1]$, the $\vec{\xi}[i+1]$ can be interpreted as the *desired* visual state after the move; it will generally be set to $\vec{\alpha}$. Learning samples obtained from $\mathcal{R}$ can be immediately used in the representation of $\mathcal{N}'$. The network $\mathcal{N}$ which is the one needed for controlling the robot represents a hyperplane in $\mathcal{N}'$. This method, introduced in (van der Smagt, Jansen, & Groen, 1992), is called **learning by interpolation**.

In the sequel, the former method will be applied. For results with the second method, see (van der Smagt et al., 1992).

5.2 Error in area

Now what is the error in the measurement of ξ_A, given an ideal lens? Firstly, due to the discretisation of the measurement of the area of the object (i.e., the projection of the continuous surface on the discrete CCD), a certain amount of error will be present. This error in a measured area ξ_A will maximally be in the order of the number of pixels on the edge of the object. This perimeter is at least $2\sqrt{\pi\xi_A}$ (for a circular object); for a square object, it is $4\sqrt{\xi_A}$. With the assumption that the noise is Gaussian, this maximum error is approximately equal to 3σ.

To investigate this assumption, a series of tests were performed where the robot arm is kept at a constant position relative to the stationary object. An image $\mathcal{I}$ was thresholded using a fixed threshold and the number of white pixels was counted. A typical probability density for the measured area of a blob is shown in figure 10a. The standard deviation in the area measurement increases linearly with the circumference, as can be seen from the measurements in figure 10b. Note that the standard deviation is much smaller than the worst-case $\sigma = 2/3\sqrt{\pi\xi_A}$. A model for



a. b.

Figure 10: a. Distribution of the measured area for a single, stationary circular blob. b. Average circumference (square root of area) and standard deviation of the measured area for several blob sizes. In all tests, a sample size of 5,000 was used.

the error in the perimeter is the following. Due to the noise, a part of the perimeter with a width of Δr_1 pixels will disappear, i.e.,

$$\Delta r_1 2\sqrt{\pi \xi_A} = \mathrm{sd}(\xi_A). \tag{13}$$

From the measurements, it is found that $\mathrm{sd}(\xi_A) \approx 0.036\sqrt{\xi_A}$ such that $\Delta r_1 \approx 0.010$ pixels.

A second error will be due to focusing. In the optical system used, the image plane (CCD) is placed at the focal distance f, such that the point of focus is at infinity. An object placed at distance d from the lens will be projected on a point h from the lens, such that

$$\frac{1}{d} + \frac{1}{h} = \frac{1}{f}. \tag{14}$$

Since, from figure 11,

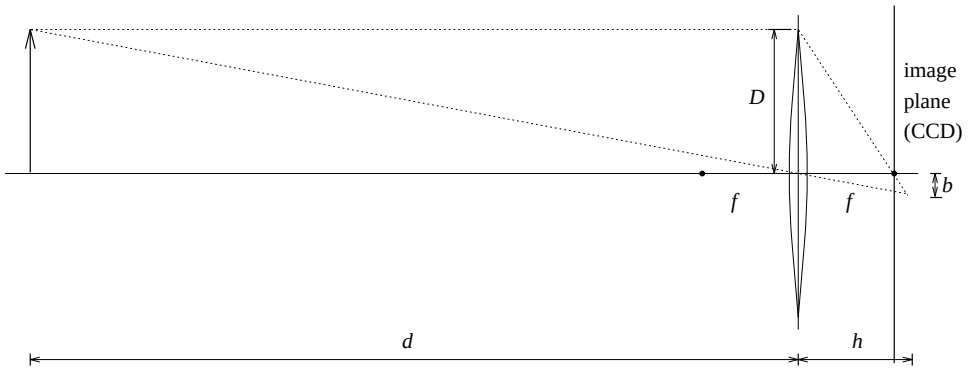


Figure 11: An optical system with the CCD placed at focal distance from the lens. The projected point is out of focus, resulting in a blur radius b.

$$\frac{D}{h} = \frac{b}{h - f} \tag{15}$$

where D is the aperture of the lens, we find that the **blur radius** b is

$$b = \frac{D(h - f)}{h} = \frac{Df}{d}. \tag{16}$$

For the camera used in the experiments, the values of the variables are $f = 4.8$mm and $D \approx 1$mm (according to definition, the aperture D is related to the f-number $f/\#$ as $D = f/(f/\#)$; in the experiments, $f/\# \approx 5$ was used). This focusing problem can be safely ignored if the b is less than pixel size. For the camera used, the pixel size is in the order of 15μm $= 0.015$mm (robot control with the nested network approach, 1984); substituting this for b, we obtain a 'safe' distance of $d \approx 0.3$m. In reality, the camera is used up to a distance of approximately 10cm, corresponding with up to 3 pixels of blur. However, at such close ranges this error can be ignored due to the larger observed object size. The relation between the blur and the area is the following. The observed area ξ_A (pixel^2) is related to the real area ξ_{A_0} (pixel^2), the focal length f (mm), and the distance d (mm) as

$$\xi_A = \xi_{A_0} \frac{f^2}{d^2}. \tag{17}$$

For the blob used in the experiments, this results in $d \approx 6{,}000/\sqrt{\xi_A}$mm. Substitution in (16) gives

$$b \approx 0.0008\sqrt{\xi_A}\mathrm{mm} \approx 0.053\sqrt{\xi_A}\mathrm{pixels}. \tag{18}$$

Similar to above, we find a theoretical upper limit for $\Delta r_2 = 0.015$ pixels.

5.3 Error in arm positioning

The precision of the robot is tested by moving it randomly at various joint speeds, and checking the resulting position against the desired position. For each joint speed, 100 trials are performed. The result of this test is shown in figure 12. The construction of joint 5 explains the better

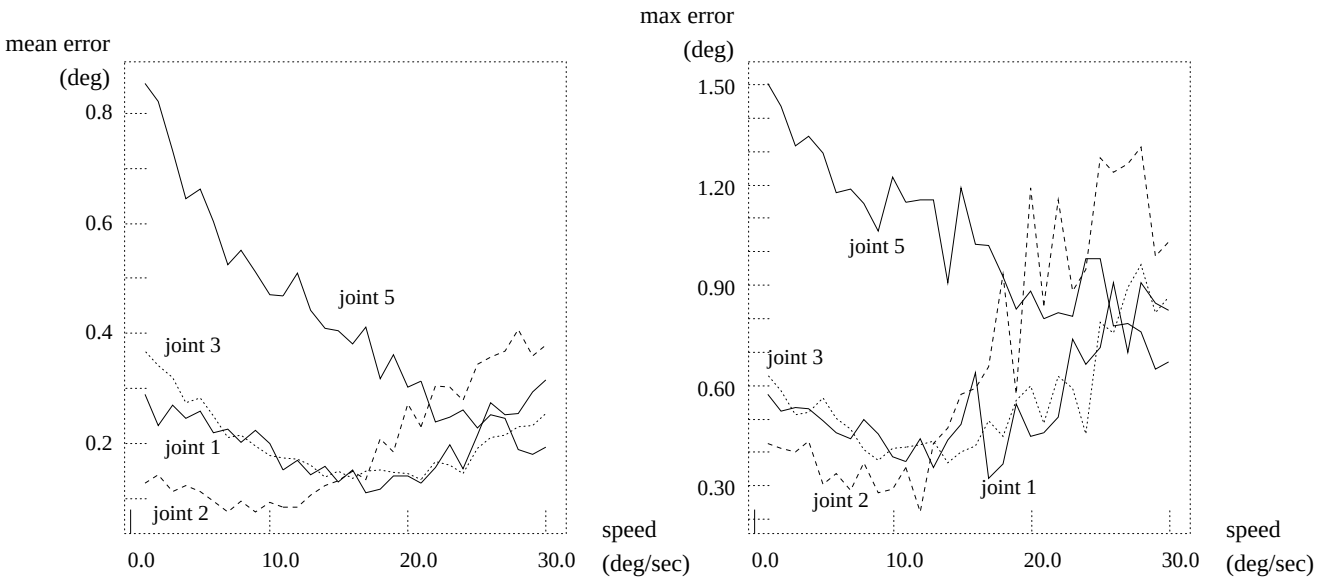


Figure 12: Precision of PI controller at various joint speeds.

positioning behaviour for this joint at higher joint velocities: it is expected to suffer mostly from stiction (NXA1010, 1992). Stiction is mostly a problem at low velocities.

Most important is the error in θ_5. In the presented implementation, θ_5 is not a parameter of the system. Therefore, the rotation of the image plane due to an error ε_5 in the rotation θ_5 is not measured and corrected for. Figure 13 shows the cause of the error. The centre of the camera

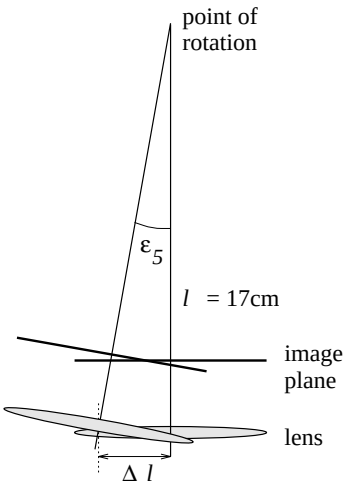


Figure 13: Translation and rotation of the image plane due to error in θ_5.

image is shifted over a distance $\Delta x = (z + 17\text{cm})\cos\varepsilon_5$ over the table; the constant 17cm is the

distance from the lens to the point of rotation. For $\varepsilon_5 \approx 0.5^\circ$ (see figure 12) and $z = 50$cm, the shift on the table is $\Delta x \approx 12$mm. With table 1, this corresponds to a rotation of approximately 1° in the joints.

The errors in joint values influence the grasping precision. Obviously, requesting a precision of much more than 1 degree per joint will be difficult to attain, since the robot cannot reach that accuracy. In summary, an accuracy exceeding 1° per joint is not to be expected with the robot used.

6 Structure of the control system

In this section, the details of the implementation are discussed. A description of the input/output behaviour, as well as a typical control loop, is given.

6.1 The neural network

The neural network consists of a feed-forward network trained with conjugate gradient back-propagation (van der Smagt, 1993a). The visual inputs $\vec{\xi}$ and robot state inputs $\vec{\theta}$ together make 5 network inputs (the structure of the robot eliminates the use of θ_1 as network input); three outputs ϑ_1, ϑ_2, and ϑ_3 constitute the network output and are given to the robot as a rotation (delta joint value) for joints 1, 2, and 3. The visual input consists of the position (ξ_x, ξ_y) measured in pixels relative to the camera centre, plus the area ξ_A of the object, also measured in pixels. With specialised hardware these quantities are measured in real time, i.e., within the time needed for grabbing one frame. At the time a frame is grabbed, and before it is operated upon by the blob finder, the robot position is measured and stored in a register. When new visual and robot data is measured, the synchronised measurements are sent. Thus it is ensured that robot and camera data are synchronously measured.

In accordance with the experiments described in (Vyšniauskas, Groen, & Kröse, 1992), to obtain 1° accuracy in joint positioning a network with 27 hidden units in a single hidden layer was used in all experiments.

6.1.1 Learning in the network

The conjugate gradient learning method is computationally expensive. To prevent delays in the forward sweep through the neural network when it is used for controlling the robot, the neural network is implemented as a parallel algorithm, consisting of a learning part (called 'lummel') and a controlling part (called 'connel'). Each of the two parts runs on a different machine, in this case a Sun workstation. The interrupt-driven communication between the two parts consists of learning samples being sent from connel to lummel, and weight matrices in the opposite direction.

Optimisation with the conjugate gradient learning method is done over a set of learning samples. In order to calculate the gradient of the weights in terms of the error in the learning samples, all learning samples that contribute to the approximation need to be examined. It is therefore chosen that a uniform representation of the learning samples is maintained by means of a multi-dimensional bin (c.q., a hypercube) in which the learning samples are stored. The dimensionality of this bin equals the number of network inputs (i.e., 5), such that the bin can be seen as a discrete representation of the input space. When a learning sample arrives from connel, the corresponding bin is located and the sample is cast into it. In the meantime, the conjugate gradient learning algorithm continuously performs minimisation over the learning samples which are stored in the bins. As soon as a request for sending a weight matrix is received, the most recent weight matrix is sent to connel.

Since a higher positioning accuracy is needed when the observed object position is close to $\vec{\alpha}$, the $\vec{\xi}_x$, $\vec{\xi}_y$, and $\vec{\xi}_A$ axes of the bin have a higher density there. Along these axes, the bin size (i.e., upper and lower limits for the bin) increases exponentially with the distance to 0. In the implementation, 5^5 bins were used. For the exponential axes, the bin size related approximately

as 14:4.7:1.7:4.7:14 (these values were obtained from an algorithm with logarithmic calculations based on freely chosen constants, expressing 'large':'medium':'small':'medium':'large').

6.2 Feedback loop

The control system is used in a feedback loop, as shown in figure 4c. Using a feedback loop also has a practical advantage for controlling a hand-in-eye system: if the system were used in a forward control loop, it would be very likely that the object is lost out of sight. This means that no learning samples can be made, so no further control information is present to make a second, more successful, attempt.

From measurements obtained at steps i and $i+1$ it is derived that the joint rotation from $\vec{\theta}[i]$ to $\vec{\theta}[i+1]$ corresponds with an object translation from $\vec{\xi}[i]$ to $\vec{\xi}[i+1]$. Using the input adjustment method, a learning sample $(\vec{\theta}[i], \vec{\xi}[i] - {}^{i}R_{i+1}\vec{\xi}[i+1]; \vec{\theta}[i+1] - \vec{\theta}[i])$ is constructed. By using the sequence of steps, not only this learning sample can be constructed, but also the samples $(\vec{\theta}(i-i_\rho), \vec{\xi}(i-i_\rho) - {}^{i-i_\rho}R_{i+1}\vec{\xi}[i+1]; \vec{\theta}[i+1] - \vec{\theta}(i-i_\rho))$ for $1 \leq i_\rho \leq i$. In figure 7, this means that also the step from $\vec{\theta}_0$ to $\vec{\theta}_2$, $\vec{\theta}_0$ to $\vec{\theta}_3$, $\vec{\theta}_1$ to $\vec{\theta}_3$ etc. are used for generating learning samples. All resulting learning samples are sent to lummel. Depending on whether the object was reached (i.e., $\vec{\xi}[i+1] \approx \vec{\alpha}$), the learning samples are marked 'perfect' or 'imperfect'; when accepted by lummel, perfect learning samples can replace imperfect ones in the bin, but never the other way around. Subsequently, the next feedback step is entered for $i+2$. The feedback loop is depicted in figure 14.

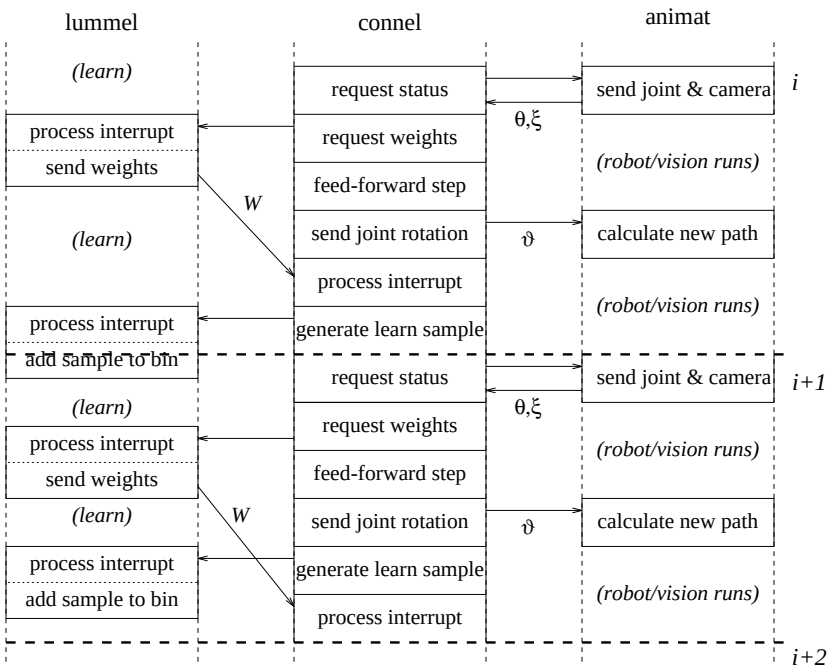


Figure 14: Schematic depiction of the steps performed in the feedback loop.

It is essential to note that the $\vec{\vartheta}[i]$'s are not used in the construction of the learning samples, but rather the real rotation done by the robot between times i and $i+1$. The $\vec{\vartheta}[i]$'s which are received by the robot are interpreted as new setpoints to which the robot must move. In general, a setpoint $\vec{\vartheta}[i]$ will not be reached at the time a new setpoint $\vec{\vartheta}[i+1]$ is received by the robot. The motion plan (i.e., set of via-points) will be adapted such that the robot moves from the current position $\vec{\theta}$ to the target position $\vec{\theta} + \vec{\vartheta}[i+1]$.

The implementation of the controller is shown in figure 15. The feedback loop is used while

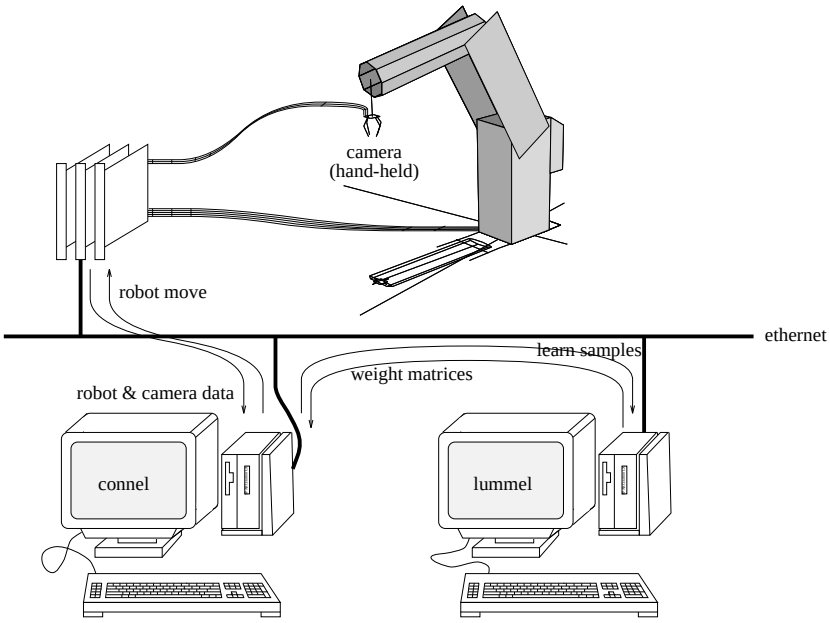


Figure 15: Implementation of the controlled system.

the robot is in motion. At step $i+1$, connel sends requests for $\vec{\xi}[i+1]$ and $\vec{\theta}[i+1]$ from the vision and robot software, and waits for the result. Next, connel sends a weight matrix request to lummel (the reply of which usually arrives one feedback step later) and performs a feed-forward sweep through the network with the weight matrix received at an earlier time. The resulting joint rotation $\vec{\vartheta}[i+1]$ is sent to the robot.

7 Results

The results of the learning must be expressed in the grasping precision of the robot. However, due to the feedback loop and the repeated interruption of moves, it is not possible to report the error after one, two, ... steps.

By using a simulator, it is also possible to let the robot finish each move. Thus the true error in the end-effector position can be reported in simulation experiments.

7.1 Simulated robot

For the simulated system, the error bound of 1° is not relevant. Positioning of the robot arm is exact, and the error reported is only due to the neural controller(*). Since all positioning data is available, it is possible to measure the Euclidian distance between the end-effector and the target object.

In order to test the accuracy of the neural controller, in the simulated environment the feedback loop is only used to make a second, third, etc. move towards the target, after the previous move has been finished. The tests are performed as follows:

1. the target object is placed at a random position in the reach space of the robot. The reach space used in the experiments is depicted in figure 16;

2. the robot is given a random initial joint position, such that the end-effector resides in the reach space;

(*) A description of the robot simulator is given in (van der Smagt, 1993b).

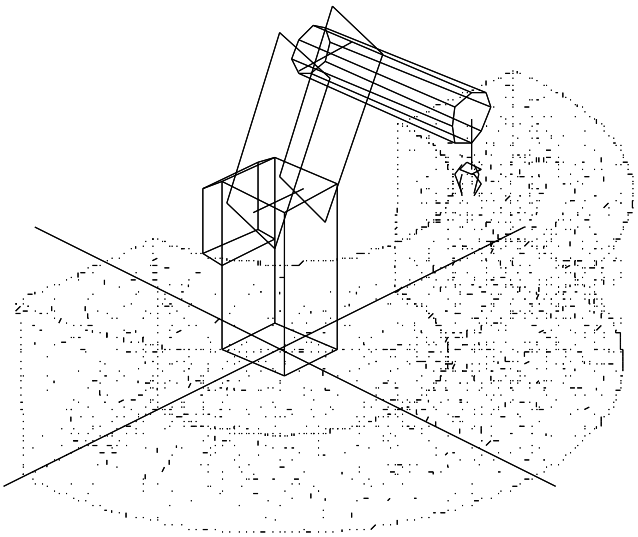


Figure 16: Reach space of the simulated robot used in the experiments.

3. the learning algorithm is started, where the robot finishes each move. The algorithm is continued until the distance between the end-effector and the object is not more than 0.5cm, or too many steps are made;

4. back to 1.

Since the feedback loop is not used as such, the functions that are approximated by the neural controller are, in essence, the robot's inverse kinematics using an egocentric viewpoint.

The results are plotted in figure 17. When compared with other methods, learning in this algorithm is very fast. By applying a topology conserving neural network for a very similar task, i.e., hand-eye coordination, Ritter *et al.* report a positioning accuracy of approximately 0.5cm after two goals, but need in the order of 5,000 learning goals to obtain this result (Ritter et al., 1989; Ritter, Martinetz, & Schulten, 1990). Results, when run on the monocular OSCAR simulator, are the same (Jansen, van der Smagt, , & Groen, 1993). The topographic map used by Kuperstein (Kuperstein, 1988) needs approximately 4,000 trials for convergence.

7.2 Results with OSCAR

For the OSCAR robot, it is not possible to measure the Euclidian distance between object and end-effector, since not only is the position of the object unknown, but such a calculation would also include uncertainties in the available robot model. Also, as explained above the feedback loop makes such an evaluation of the algorithm impossible. Therefore it is chosen to evaluate the difference between the final reached position in joint space, and the previously predicted positions during the feedback loop. This is therefore an error in the setpoints ϑ_i for joint i:

$$\sum_{j=0}^{n_\tau} |\vartheta_i[j] + \theta_i[j] - \theta_i[n_\tau]| \tag{19}$$

This integral gives the average predicted error in joint space for one trial (i.e., one target position) (see figure 18). The number of imperfect and perfect learning samples which are stored in the bin maintained by lummel is also shown in figure 18. The number of perfect learning samples grows asymptotically towards the optimal number as found in (Vyšniauskas et al., 1992); the same steepness is encountered in the error curve 18 (left), indicating that a stable solution has been found.

8 Discussion

From the chosen number of hidden units, and the number of learning samples that is effectively stored in the bins, according to (Vyšniauskas et al., 1992), an error of approximately 1° per joint, or approximately 2.5cm in the end-effector is expected. This error is indeed found in curves 17. Using the same theory, the desired precision can be adapted by increasing the number of learning samples and hidden units.

The question arises whether the feed-forward network with sigmoidal hidden units is the optimal choice for representing the inverse kinematics plus the camera-hand mapping. Knowledge of the analytical form of the approximated functions may assist in a better choice of basis functions: it is not optimal to approximate trigonometric functions by means of sigmoids. However, the goal is to control the real robot rather than the simulated one. Knowledge of the inverse kinematics of the simulated robot does not include the errors due to the dynamics, and the camera deformations. It is therefore not preferable to use more modeling in the approximation.

In (van der Smagt et al., 1992; Jansen et al., 1993) the global approximation, as presented in this paper, is combined with local approximations (Ritter et al., 1989), by building a multi-resolution approximation of the function in its input space. Though computationally expensive, this approach has been shown to be very accurate while showing fast (i.e., based on few learning samples) learning. Investigation in this direction is being continued.

Reference

Åström, K. J., & Wittenmark, B. (1989). *Adaptive Control*. Addison-Wesley Publishing Company.

Ballard, D. H., & Brown, C. M. (1982). *Computer Vision*. Prentice Hall, Inc.

Bartholomeus, M. (1993). Vision. , Universiteit van Amsterdam.

Bellman, R., & Kalaba, R. (1959). On adaptive control processes. *IRE Transactions on Automatic Control*, *4*, 1–9.

Craig, J. J. (1986). *Introduction to Robotics*. Addison-Wesley Publishing Company, Reading, MA.

Feldbaum, A. A. (1965). *Optimal Control Systems*. Academic Press.

Fu, K. S., Gonzalez, R. C., & Lee, C. S. G. (1987). *Robotics: Control, Sensing, Vision, and Intelligence*. McGraw-Hill Book Company, New York.

Jansen, A., van der Smagt, P. P., , & Groen, F. C. A. (1993). Nested networks for robot control. In Murray, A. F. (Ed.), *Neural Network Applications*. Kluwer Academic Publishers. in print.

Kanade, T. (1981). Recovery of the three-dimensional shape of an object from a single view. *Artificial Intelligence*, *17*, 409–460.

Kröse, B. J. A., van der Korst, M. J., & Groen, F. C. A. (1990). Learning strategies for a vision based neural controller for a robot arm. In Kaynak, O. (Ed.), *IEEE International Workshop on Intelligent Motor Control*, pp. 199–203. Istanbul.

Kuperstein, M. (1988). Neural model of adaptive hand-eye coordination for single postures. *Science*, *239*, 1308–1311.

Lagerberg, J., van Albada, D., & Visser, A. (1993). ESPRIT 2 Project CAR (5220).

Marr, D. (1982). *Vision*. W. H. Freeman and Company.

Narendra, K. S., & Annaswamy, A. M. (1989). *Stable Adaptive Systems*. Prentice Hall, Inc.

Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

OUG (1992). Oscar user's meeting..

Philips (1984). *Development Sample Data: Frame Transfer Sensor NXA1010*.

Psaltis, D., Sideris, A., & Yamamura, A. A. (1988). A multilayer neural network controller. *IEEE Control Systems Magazine*, *8*(2), 17–21.

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

Ritter, H. J., Martinetz, T. M., & Schulten, K. J. (1990). *Neuronale Netze*. Addison-Wesley Publishing Company.

van der Smagt, P. (1993a). Minimisation methods for training feed-forward neural networks. *Neural Networks*. in print.

van der Smagt, P. (1993b). Simderella user's guide. , Universiteit van Amsterdam.

van der Smagt, P., Jansen, A., & Groen, F. (1992). Interpolative robot control with the nested network approach. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 475–480. Glasgow, Scotland, U.K.

Vyšniauskas, V., Groen, F. C. A., & Kröse, B. J. A. (1992). Function approximation with a feed-forward network: the optimal number of learning samples and hidden units. , Department of Computer Systems, University of Amsterdam, Amsterdam, Netherlands.

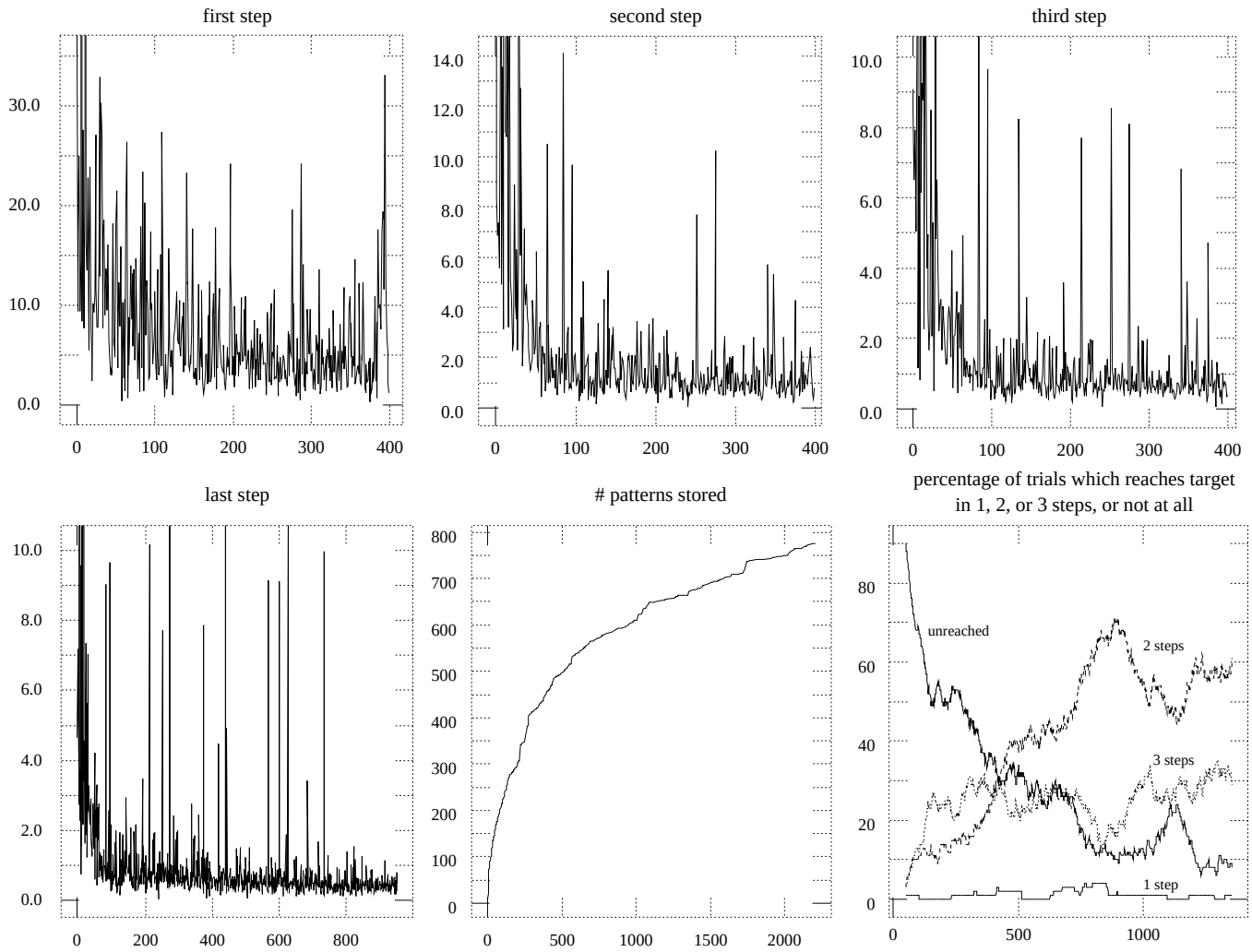


Figure 17: **Top row.** Grasping error in cm with the simulated robot after 1, 2, and 3 steps. The desired accuracy was a Euclidean distance of 0.5cm between the end-effector and the target object. **Bottom row.** Left: final grasping error. Centre: number of stored learning samples. Right: percentage of runs in which 1, 2 and 3 steps were needed to reach the target; percentage of runs in which the target was not reached within 0.5cm. This data was smoothed with a moving average filter of width 100.

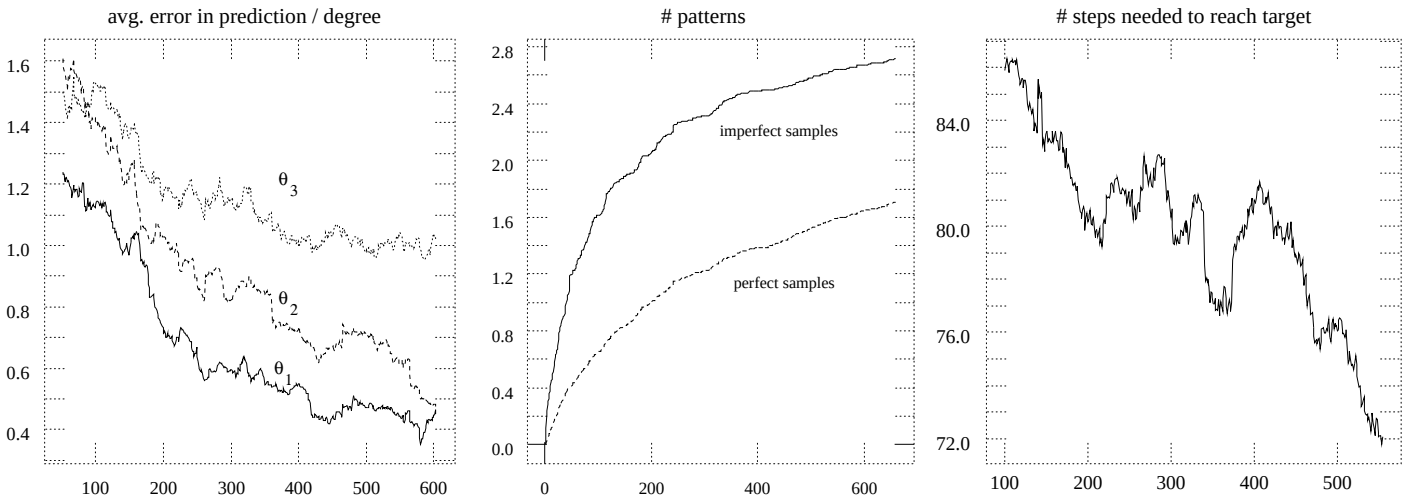


Figure 18: (Left) Error in generated $\vec{\vartheta}$ depicted for joints 1, 2, and 3. The data is smoothed with a moving average (MA) filter of width 100. (Centre) Number of perfect and imperfect learning samples stored in the bin. (Right) Number of steps between the beginning of the grasping and the moment when the object is reached. The data is smoothed with a 200 MA filter.