Teaching a robot to see how it moves

Patrick van der Smagt
Institute for Robotics and System Dynamics
German Aerospace Research Establishment (DLR)
P. O. Box 1116, 82234 Wessling, GERMANY
URL `http://www.op.dlr.de/FF-DR-RS/`
email `smagt@dlr.de`

November 7, 1997

1 Introduction

The positioning of a robot hand in order to grasp an object is a problem fundamental to robotics. The task we want to perform can be described as follows: given a visual scene the robot arm must reach an indicated point in that visual scene. This marked point indicates the observed object that has to be grasped. In order to accomplish this task, a mapping from the visual scene to the corresponding robot joint values must be available. The task set out in this chapter is to design a self-learning controller that constructs that mapping without knowledge of the geometry of the camera-robot system.

When the position of the object is unequivocally known with respect to the robot's base, and the robot geometry (the kinematics) is known, a single computation followed by a robot joint rotation suffices to reach the indicated position.

But what if these data are unavailable or, a more typical case, are too inaccurate to solve the problem and grasp the object? To tackle this problem, we assume an academic problem and let go of any explicit model of the robot or its sensors. So, we assume that the visual system needs no precise calibration. The solution is given by *learning*: a neural controller has to learn to generate robot joint rotations which position the end-effector directly above the target object. Figure 1 demonstrates the task. Note that, due to the fact that the designed system does not rely on a model of the robot, it can in fact be applied to *any* robot arm.

In order to design a system which can be successfully used in real-world applications, there are two important issues which have to be considered. First, in real-world applications of robot systems, a 'reasonable' training time must be ensured. Real robots move slowly in comparison with simulated robots, so it is important that after only a few trials the goal is reached. Hundreds of trials are not acceptable. Secondly, the added value of self-learning systems must be fully exploited: it is essential that the method adapt to unforeseen gradual or sudden changes in the robot-camera system. The combination of these two points has been ignored in many previous approaches.

The system is thus set up that the relation between sensor input and robot motion 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

[0]In: *Neural Network Perspectives on Cognition and Adaptive Robotics*, ed. Antony Browne, Institute of Physics Publishing, 1997, pp. 195–219.

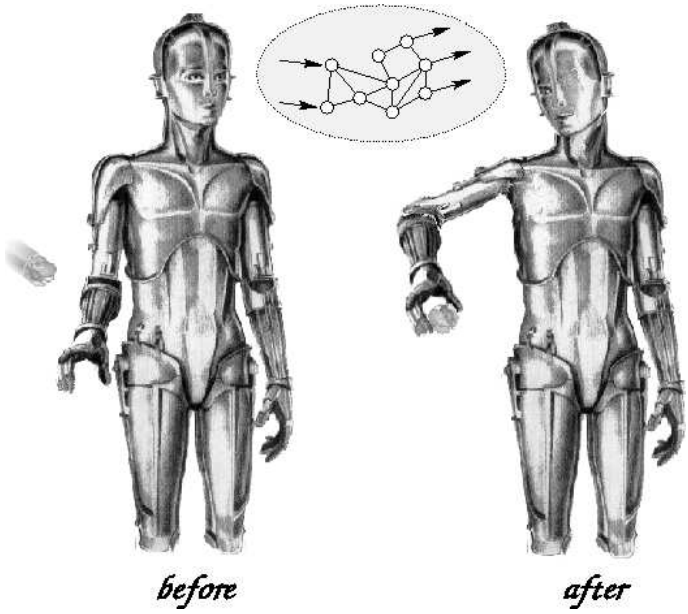

Figure 1: The task: grasp an object.

reach a specific position; for instance, there is no elbow up–elbow down ambiguity. As customary in robotics, this situation is realised by choosing a preferred situation (c.q., the elbow up configuration) and never presenting 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. In practice, however, it is never learned and therefore does not occur.

In our approach no explicit models of the camera or the robot are available. The camera-hand mapping must be learned by the neural network based on the (measured) behaviour of the camera-robot system. Hereto **learning samples** are gathered during the control process and added to the **learning set**. The size and exact implementation of this learning set is directly related to the adaptability and accuracy of the system: a large learning set leads to a sluggishly adapting system, whereas a small learning set cannot be used to construct an accurate controller. Therefore its size must be varying.

Summing up, we have to tackle the following four problems:

1. how do we control the robot without having a model of the robot nor of its sensors?
2. how do we get the (computationally intensive) neural network to learn the robot's behaviour on-line?
3. how do we ensure that the resultant approximation is precise enough to control the robot in its large reach space?
4. is the neural network fast enough for real-time robot control while handling visual data?

To understand what these requirements mean we first need to understand some of the pitfalls of robotics and vision.

2 The components

2.1 Robotics

In robotics, when we restrict ourselves to the control of robot arms, we are faced by three problems:

1. assuming that a target position is known (in Cartesian or sensor space) where the hand of the robot arm must go to, a set of joint angles must be computed with which the robot can reach that position. This problem is known as *inverse kinematics*.
2. Secondly, a path must be generated along which each of the joints must be moved in order to reach the target position from the current position. This problem is known as *path planning*.
3. Third, and finally, the right forces must be exerted on the joints (e.g., by giving the motors of the robot the right currents) in order to actualise the motion. This problem is known as *inverse dynamics*.

Computation of the inverse kinematics is a solvable problem, provided that the dimensions of the robot are known. This knowledge depends on a model of that robot, and of course a problem exists when a model is not available or, which is often the case, not very accurate. One should realise that an error of a fraction of a degree in robot arm rotation can result in a few mm up to a cm of positioning error of the robot hand. The model therefore has to be very precise.

More complex are the inverse dynamics. The dynamics of any d degree of freedom robot with rotational joints can be described by the equation (Craig, 1986)

$$T\left(\theta,\dot{\theta},\ddot{\theta}\right) = F_1(\theta)\ddot{\theta} + F_2(\theta)\left[\dot{\theta}\dot{\theta}\right] + F_3(\theta)\left[\dot{\theta}^2\right] + F_4(\theta,\dot{\theta}) + F_5(\theta) \tag{1}$$

where T is a d-vector of torques exerted by the links, and θ, $\dot{\theta}$, and $\ddot{\theta}$ are d-vectors denoting the positions, velocities, and accelerations of the d joints. $[\dot{\theta}\dot{\theta}]$ and $[\dot{\theta}^2]$ are vectors

$$\left[\dot{\theta}\dot{\theta}\right] = \left[\dot{\theta}_1\dot{\theta}_2, \dot{\theta}_1\dot{\theta}_3, \ldots, \dot{\theta}_{d-1}\dot{\theta}_d\right]^T, \qquad \left[\dot{\theta}^2\right] = \left[\dot{\theta}_1^2, \dot{\theta}_2^2, \ldots, \dot{\theta}_d^2\right], \tag{2}$$

$F_1(\theta)$ is the matrix of inertia, $F_2(\theta)$ is the matrix of Coriolis coefficients, $F_3(\theta)$ is the matrix of centrifugal coefficients, $F_4(\theta,\dot{\theta})$ is a friction term, and $F_5(\theta)$ is the gravity working on the joints.

When the robot has to move from one joint position to another, a torque must be applied which generates T. The problem of calculating the correct torques (forces) to have the robot arm follow a specified trajectory is known as **inverse dynamics**. Industrial robots are generally designed to eliminate the interdependence between the joints, such that the robot arm can be regarded as d independent moving bodies. In that case, F_1 and F_3 are diagonal matrices and F_2 is zero. This reduces the $3d$-values vector field as described in (1) to d independent functions of three variables for which the coefficients have to be found. Also, the link actuators are usually made so powerful that F_1, F_3, F_4, and F_5 can be considered independent of θ. For this simplified (and common) case, various standard methods exist to compute the inverse dynamics (Fu, Gonzalez, & Lee, 1987). This controller eliminates the requirement of knowledge of the robot arm in order to control it. When using such a control method, the robot can be controlled by specifying joint values, velocities, and accelerations, and knowledge of the required forces is not required. However, this control method requires a precise model of the robot, which may not be available. Adaptation and learning is therefore an important tool.

2.2 Vision

The vision problem is different: here we are faced with huge amounts of data (in particular, a black-and-white camera generates approximately 0.25Mb of data each 40msec (or over 6Mb of data per second); a colour camera three times as much). How are we going to handle this data?

Clearly, even the fastest workstation, which can make over 100Mflops (a *flop* is a single floating-point operation) per second, has no chance. Although 6 million of multiplications can be done in about 0.4s, when in the meantime 6 million memory locations have to be accessed this time can go up to more than 10s.

Therefore it is wise to *pre*process the visual data. With specialised hardware, we might do the following tricks in real-time (i.e., as fast as the images are generated):

- reduce a grey-level image to black-and-white (2-colour);
- subsample the b&w image, i.e., reduce a full-size 512×512 image to 256×256 or 64×64 or .... In fact, reduction up to 32×32 is possible without losing data; after all, the binary b&w image needs only one bit per pixel!
- select all the regions which are white and compute their position and orientation in the image (blob finding and moment computation).
- select the object that we are really interested in, and find a unique form for its position (3 integer values) and orientation (3 floating point values).

Thus the 0.25Mb per 40msec can be reduced to a few bytes only. These data can be easily processed by a neural network or any other controller.

Nice at it seems, there is of course a price to be paid. The flexibility of the resultant system is limited due to the approach. As much as we would have liked to, it is just not feasible to pump the whole image into a self-learning neural system!

Naturally, there are exceptions to this rule. One of the most famous examples is the ALVINN, a neural-network based vehicle driving system. The neural network has as input a sub-sampled 30×32 pixel image of from a camera mounted on the roof of a car. The input is fully connected to a 5-unit hidden layer, which in turn is connected to 30 output units, each of which indicates a direction in which the vehicle has to steer. To teach ALVINN to steer, a driver has to drive the car for about three minutes while ALVINN is learning. Due to its generalising ability, ALVINN is not only able to drive on roads marked with white and yellow stripes, but has demonstrated its ability to stay on single-lane dirt roads, single lane paved bike paths, two lane suburban neighborhood streets, and lined divided highways. On the last domain speeds of up to 70mph were reached on public highways. Note, however, that for each road type a network must be specifically trained. The choice of which network is best used is also taken care of by ALVINN.

2.2.1 Visual setup

For the system to accomplish the specified task it must know the position of the observed object relative to the robot, in *some* coordinate system. Consider a robot that a specific moment has a joint position θ. The position of the end-effector in world coordinates is given by $\mathbf{x}_r$. The robot has to move towards an object located at world position $\mathbf{x}_o$, i.e., assume a θ such that the $\mathbf{x}_r$ equals $\mathbf{x}_o$. As discussed above, $\mathbf{x}_o$ and $\mathbf{x}_r$ are not available without an accurate model of the visual system. There are two basic visual setups to obtain the required visual information:

1. **External 'world-based' camera:** both the robot and target object are observed by cameras situated at fixed and unchanging positions. The visually observed object features (indicated by $\{\xi_i\}_o$) in the image must uniquely determine the position $\mathbf{x}_o$ of the 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 visually observed robot features $\{\xi_i\}_r$ representing the robot end-effector must uniquely determine its world coordinates $\mathbf{x}_r$, 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 observed object position (the visual observation ξ) may not uniquely describe the object position in world coordinates. However, now measurements ξ together with the robot position θ must uniquely define the world coordinates $\mathbf{x}_o$ if a non-ambiguous motion plan has to be made.

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 end-effector position. 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 quantised camera image is inversely proportional to the distance between the camera and the observed scene (when the focus of the optical system is fixed). Thus the visual precision increases when the target object is approached; the finite resolution of the camera is no limiting factor. 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 for a robot with a typical arm length of 1m. Thus their precision will be limited, typically 0.5–1cm for the application in this chapter.

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 controllers developed within the scope of this thesis needs such information, additional *a priori* knowledge is required. Two solutions are considered.

1. **Model-based monocular:** in this case, *a priori* knowledge of the observed object is assumed. For instance, when a sufficient set of point features of the object can be observed, and the position of these features on the object is known, the position of the object relative to the camera in world space can be reconstructed. For more detailed information about robust model-based approaches, consult, e.g., (Kanade, 1981; Lamdan & Wolfson, 1988; Gavrila & Groen, 1992). An exemplar method is the following: when the camera is looking down to the scene consisting of a single object with a flat, horizontal surface, the observed area of the object is inversely proportional to the square root of the distance.

2. **Correspondence-based binocular:** When no *a priori* knowledge is present, triangulation can be used to measure depth. This can be realised with a stereoscopic system (Ballard & Brown, 1982; Fu et al., 1987). When two cameras, whose image

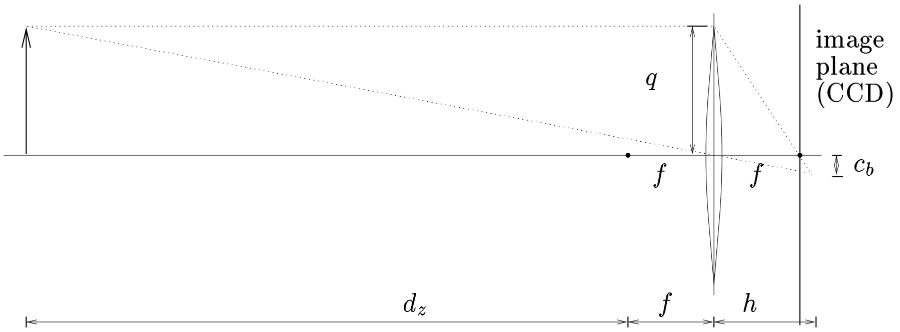

Figure 2: An optical system with the CCD placed at focal distance from the lens. The symbols are referred to in the text.

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 a significant problem: which point features in the left image correspond with which point features in the right image?

2.2.2 Implementation

In conclusion, the advantage of model-based monocular vision is the increasing precision, avoidance of the correspondence problem, and simpler image processing. An additional advantage is that the problem of occlusion of marker points or parts of the observed object (e.g., due to the rotating robot arm) is avoided. A disadvantage of this method is the requirement of *a priori* knowledge of the observed scene.

Relative depth information can be obtained by using sequences of visual images. By measuring the divergence in an image when approaching an object (e.g., how much an observed object gets larger when it is approached), the visual distance divided by the visual velocity. Note that the absolute distance cannot be measured; e.g., a large object far away cannot be distinguished from a nearby small object.

In most CCD cameras, the image plane (CCD) is placed at the focal distance f from the lens, such that the point of focus is at infinity. An object placed at distance $d_z + f$ from the lens will be projected on a point h from the lens, such that (Hecht & Zajac, 1974)

$$\frac{1}{d_z + f} + \frac{1}{h} = \frac{1}{f} \tag{3}$$

as depicted in figure 2. In the proposed system, the depth d_z will be derived from the projected (or observed) area ξ_A with relation to the 'real' area A of the object; A is defined as the projected area of the object when $d_z = f$. This observed area is measured as the number of white pixels (constituting the object) on the CCD. Assuming a pinhole camera, the d_z and ξ_A are related as

$$d_z = f\sqrt{\frac{A}{\xi_A}}. \tag{4}$$

Note that f and A are constants for one particular lens and object.

Given the placement of the camera, and the data that are 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 the basic principles of (dis)continuity (i.e., edge-based) and homogeneity (i.e., region-based);

3. **image description**: for the purpose of component recognition and for subsequent use in the control algorithm, for each component, features $\boldsymbol{\xi}$ describing the properties of the component as well as its position are determined;

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

The visual system used for the experiments described in this chapter needs only identify the 3D position of an observed object; the depth is calculated from the area of the object.

2.3 Control systems

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 process**;

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

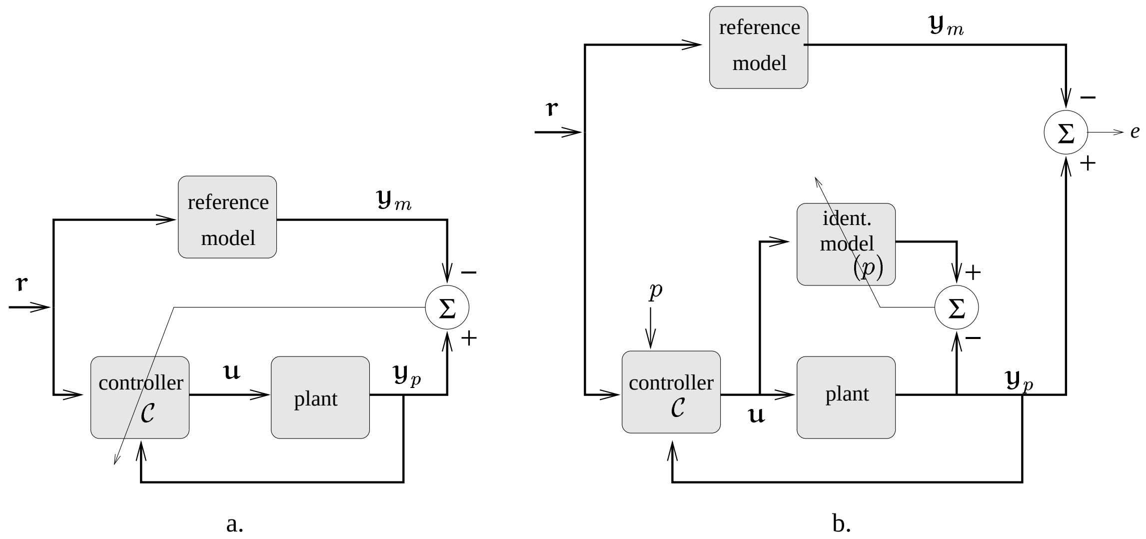

Figure 3: a. Direct adaptive control. b. Indirect adaptive control.

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 3b), the parameters of the plant (i.e., the system that is controlled) 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 3a).

The direct control method in figure 3a works as follows. The plant is controlled by a signal $\mathbf{u}$, and outputs a signal $\mathbf{y}_p$. The **reference signal** is a signal $\mathbf{r}$ which is input to the controller; $\mathbf{r}$ can be regarded as the *desired situation* or *desired state* of the system. The **reference model** is used to compute the desired plant output $\mathbf{y}_m$ from the reference signal or *setpoints* $\mathbf{r}$. The reference model translates $\mathbf{r}$ to the domain of $\mathbf{y}_p$, resulting in $\mathbf{y}_m$. The task for the controller $\mathcal{C}$ is to generate a signal $\mathbf{u} = \mathcal{C}(\mathbf{r})$ which minimises $\|\mathbf{y}_m - \mathbf{y}_p\|$, i.e., it minimises the difference between the actual and the desired situation. 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 $\|\mathbf{y}_m - \mathbf{y}_p\|$ carries information on how the output of the *plant* must change, and not the output of the *controller*. In order to adapt the controller, however, the error must be available in terms of $\mathbf{u}$. This can only be determined when $\partial\mathbf{u}/\partial\mathbf{y}_p$ is known, which requires an inverse model of the plant. This is a serious drawback of this approach, since this inverse model is usually not available.

A solution is given by 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.

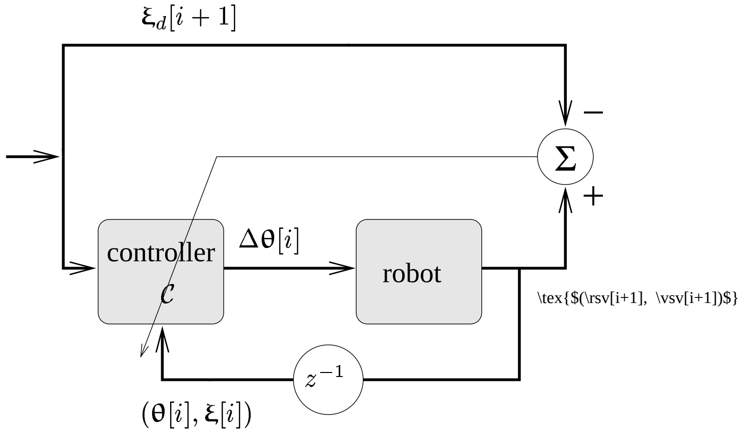

Figure 4: Structure of the proposed controller. After receiving input $\xi[i]$, $\theta[i]$, and the desired $\xi_d[i+1]$, the controller has enough information to determine the plant input $\Delta\theta[i]$. The z^{-1} is a delay.

2.3.1 Structure of the proposed controller

However, the direct control method can still be used when no reference model is needed. This is the case when the reference signal $\mathbf{r}$ is expressed in quantities which can directly be measured from the plant, i.e., $\mathbf{r}$ and $\mathbf{y}_p$ are in the same domain.

The robot/camera system can be seen as a discrete state machine, where the controller $\mathcal{C}$ is assumed to be delay-free, and we write $\xi_d[i+1]$ instead of $\mathbf{r}$, and $(\theta[i+1], \xi[i+1])$ for $\mathbf{y}_p$. From i to $i+1$ two measurable transitions occur: from $\theta[i]$ to $\theta[i+1]$ and from $\xi[i]$ to $\xi[i+1]$. Both are representations of the state of the camera-robot system; the θ the internal robot state (viz. joint values) and the ξ the visual observation. Since the camera is hand-held and only observes the object, neither representation alone suffices to uniquely describe the state of the system; this state is only given by pairs $(\theta[i], \xi[i])$.

Since the robot moves from internal joint state $\theta[i]$ to $\theta[i+1]$, we will denote the controller output which effectuates this move by $\Delta\theta[i]$. Also, note that $\xi[i+1]$ is the signal for which $\xi_d[i+1]$ is the desired value. This means that, when $\xi[i+1] = \xi_d[i+1]$, a 'successful' $\Delta\theta[i]$ had been generated and applied to the robot. Differently put, we know that

$$\mathcal{C}^0\left(\theta[i], \xi[i], \xi_d[i+1]\right) = \Delta\theta[i]$$

where $\mathcal{C}^0$ is the *ideal* controller. In all other cases a valid transition is still available:

$$\mathcal{C}^0\left(\theta[i], \xi[i], \xi[i+1]\right) = \Delta\theta[i]$$

even though $\xi[i+1] \neq \xi_d[i+1]$. Thus we can focus on direct control only, and do not require an identification model; after all, the plant output and the controller input are in the same domain. The resulting controller is depicted in figure 4.

The studied direct adaptive controller, used in all subsequent chapters, consists of a feed-forward neural network trained with conjugate gradient optimisation. We will refer to this neural controller by the symbol $\mathcal{N}$ instead of $\mathcal{C}$. The universal approximation capabilities of such networks allow them to be used in a direct control scheme while eliminating the requirement for a reference model.

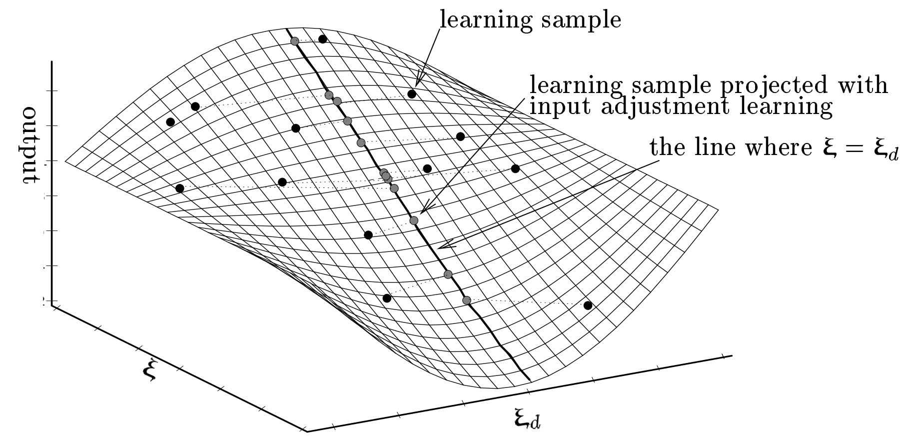

Figure 5: Learning by interpolation. All learning samples are positioned on a hyperplane (in the figure shown in two dimensions). The only part of the hyperplane that is used for control of the robot arm describes, in the figure, a line. Because of the increased dimensionality of the plane, the learning samples are distributed around the line, instead of on it.

2.3.2 Generating learning samples

The task of the network is to learn the relationship between the transition from $\xi[i]$ to $\xi[i+1]$ on the one hand, and $\theta[i]$ to $\theta[i+1]$ on the other. The neural network must be trained to generate robot motion commands $\Delta\theta[i]$ to control $\xi[i]$ towards $\xi_d[i{+}1]$. To teach the neural network from the transition $i \to i+1$ we use the **learning by interpolation** method.

In learning by interpolation (Smagt, Jansen, & Groen, 1992), besides the robot state $\theta[i]$, not only the reference signals $\xi[i]$ are input to the neural controller, but also their desired values at $i+1$, $\xi_d[i+1]$. The ideal controller $\mathcal{C}^0$ would generate a correct $\Delta\theta[i]$ which makes $\xi[i+1] = \xi_d[i+1]$. When the $\xi[i+1] \neq \xi_d[i+1]$, the generated $\Delta\theta[i]$ was not correct, but still it is known that the camera-robot system makes a transition from $\xi[i]$ to $\xi[i+1]$ which can be used as learning sample. The closer $\xi[i+1]$ and $\xi_d[i+1]$, the more useful information this transition carries. In any case, a learning sample is available: input $(\theta[i], \xi[i], \xi[i+1])$ maps on $\Delta\theta[i]$.

This process of generating learning samples is illustrated in figure 5. The learning samples will, in general, be situated 'around' the line where $\xi[i+1] = \xi_d[i+1]$, but by interpolating those learning samples, an approximation for that line is assumed to be generated.

Note that the dimensionality of the input space had been increased by the dimensionality of ξ, leading to a sparseness of the learning samples. This sparseness can be understood from figure 5. Samples are positioned on the hyperplane, whereas only the values situated on the 'hyperline' are used by the controller.

3 Using visual feedback in robot control

3.1 An experimental setup I

3.1.1 Introduction

As concluded in the previous section, the visual observation that is minimally required to grasp the object consists of the observed position (ξ_x, ξ_y) and area ξ_A of an object projected on the camera's CCD. We will write $\boldsymbol{\xi} = (\xi_x, \xi_y, \xi_A)$ to denote the visual observation. Secondly, the position of the robot is described by its joint values $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$.

In this chapter, a neural controller will be constructed which, given the state $(\boldsymbol{\xi}, \boldsymbol{\theta})$ of the camera-robot system, generates a **robot setpoint** $\Delta\boldsymbol{\theta} = (\Delta\theta_1, \Delta\theta_2, \Delta\theta_3)$, to move the system *to the goal state*, i.e., reaching the state where

$$\boldsymbol{\xi} = \boldsymbol{\xi}_d. \tag{5}$$

When this state is reached, we say that the **goal** is attained. The desired visual state is defined as $\boldsymbol{\xi}_d \equiv (0, 0, \boldsymbol{\xi}_{A,\text{desired}})$, i.e., the object must be in the centre of the camera image at the desired object size $\boldsymbol{\xi}_{A,\text{desired}}$; the latter requirement means that the distance between the camera and the observed object agrees with some pre-specified value. This visual state $\boldsymbol{\xi}_d$ is called the **visual setpoint** for $\boldsymbol{\xi}$.

Open loop control. We are trying to construct an adaptive controller which generates a robot joint rotation $\Delta\boldsymbol{\theta}$ such that, when this rotation is applied to the robot, the observed object will be located in the centre of the camera image at the predefined size. Using the symbol $\mathcal{N}$ for the neural controller mapping which we are looking for, and $\mathcal{R}$ for the given robot-camera mapping, the following two transformations are performed:

$$\mathcal{N}(\boldsymbol{\xi}, \boldsymbol{\theta}) = \Delta\boldsymbol{\theta}$$

such that

$$\mathcal{R}(\boldsymbol{\xi}, \boldsymbol{\theta}, \Delta\boldsymbol{\theta}) = (\boldsymbol{\xi}_d, \boldsymbol{\theta}')$$

where $\boldsymbol{\xi}_d \equiv (0, 0, A)$ and $\boldsymbol{\theta}' = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$. When $\mathcal{N}$ indeed generates the correct joint rotation $\Delta\boldsymbol{\theta}$ for all possible combinations of $\boldsymbol{\xi}$ and $\boldsymbol{\theta}$, we call $\mathcal{N}$ the *ideal* controller and write $\mathcal{C}^0$. So, by definition

$$\forall \boldsymbol{\xi}, \boldsymbol{\theta}: \quad \mathcal{R}\left(\boldsymbol{\xi}, \boldsymbol{\theta}, \mathcal{C}^0(\boldsymbol{\xi}, \boldsymbol{\theta})\right) := (\boldsymbol{\xi}_d, \boldsymbol{\theta}').$$

The neural controller $\mathcal{N}$ consists of a feed-forward neural network. $\mathcal{N}$ is being trained from learning samples $(\boldsymbol{\xi}, \boldsymbol{\theta}; \Delta\boldsymbol{\theta})$ from which it is known that

$$\mathcal{C}^0(\boldsymbol{\xi}, \boldsymbol{\theta}) = \Delta\boldsymbol{\theta}.$$

The method for creating these learning samples is explained in section 3.1.2. In order to find optimal weight values in the network, the Polak-Ribière conjugate gradient learning algorithm with Powell restarts is used (Smagt, 1994).

Closed loop control. However, $\mathcal{N}$ will in general not be equal to $\mathcal{C}^0$. There will always, depending on the structure of the neural network and the optimisation method chosen, remain an error in the approximation to the underlying function. It is therefore likely that, when controlling the robot arm in an open loop, the hand-held camera loses track of the object when it is approached. A single joint rotation which places the end-effector only a few cm next to the target location will make that the camera does not see the

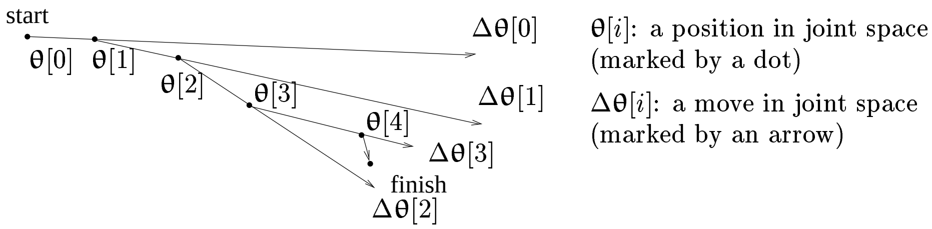

Figure 6: 2D motion plan of the robot arm. A planned move $\Delta\theta$ is, in general, not completed. While the robot is moving towards the new setpoint, a new setpoint is received, and the motion plan is updated.

object anymore because it is outside the camera's field of view, such that no information at all is available about the correctness of the previous move. By using closed loop control, thus adapting the path towards the object during the move, this problem is solved. We introduce feedback in the control loop as follows. To indicate the sequence of control, the variables ξ, θ, and $\Delta\theta$ will be time-indexed. Thus we write,

$$\mathcal{N}\left(\xi[i], \theta[i]\right) = \Delta\theta[i].$$

Now, instead of waiting for the robot to complete the move $\Delta\theta[i]$ until it is finished, *during* that move the state of the robot-camera system is measured anew and fed into the neural controller:

$$\mathcal{N}\left(\xi[i+1], \theta[i+1]\right) = \Delta\theta[i+1].$$

The resulting joint rotation $\Delta\theta[i+1]$ is sent to the robot, which immediately changes the trajectory it follows. This scheme is repeated until the target object is reached. Figure 6 illustrates the feedback control influence on the trajectory followed by the robot arm. Note that the delay between iteration i and $i+1$ is not defined; it is dictated by the visual processing and communication delays.

Why will the system be more accurate when a feedback loop is introduced? There are two reasons:

1. sensor readings close to the target are more accurate;
2. the approximation to the $\mathcal{C}^0$ close to the target can be made more accurate. This is done by increasing the sample density close to the goal state; this is automatically obtained, since a successful grasping trail always ends in this goal state, thus creating samples in that part of the input space.

The proposed experimental setup is shown in figure 7. At time i, the visual and robot state are fed into the neural network, which generates a joint rotation. That rotation is realised by the inverse robot dynamics module, which calculates torques to make the robot move. One time slot later, new visual and robot state data are available and given to the neural network.

3.1.2 Constructing the controller

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.

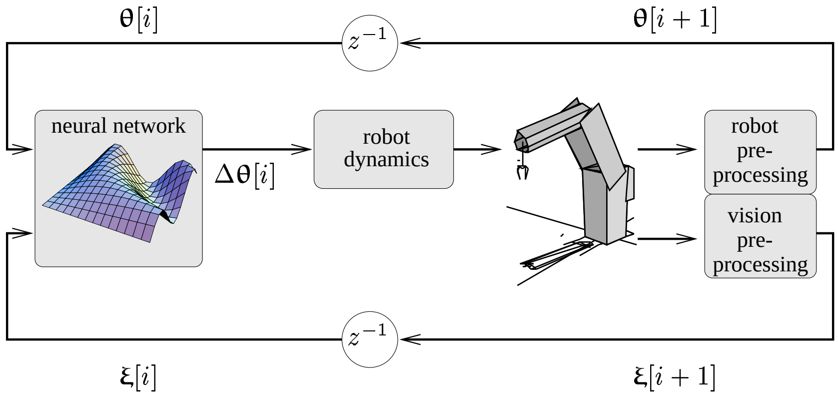

Figure 7: The experimental setup. The visual state $\boldsymbol{\xi}[i]$ and robot state $\boldsymbol{\theta}[i]$ are input to the neural controller, which generates a joint rotation $\Delta\boldsymbol{\theta}[i]$. The robot's PID controller (marked 'robot dynamics' in the figure) calculates the required torques to make the robot move. At the next step $i+1$, the new robot and visual state are available.

The neural network. The neural network consists of a feed-forward network trained with conjugate gradient back-propagation as described in (Smagt, 1994). The visual inputs $\boldsymbol{\xi}$ and robot state inputs $\boldsymbol{\theta}$ together make 5 network inputs (as discussed before, the structure of the robot eliminates the use of θ_1 as network input); three outputs $\Delta\theta_1$, $\Delta\theta_2$, and $\Delta\theta_3$ 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 the position and area of the object are extracted, the robot position is measured and stored in a register.

The neural network must learn the relationship between displacements in the visual domain and displacements in the robot domain. This relationship is contained in the measured visual data $\boldsymbol{\xi}[i]$ and $\boldsymbol{\xi}[i+1]$ in relation to the robot data $\boldsymbol{\theta}[i]$ and $\boldsymbol{\theta}[i+1]$.

Knowing that a robot move from $\boldsymbol{\theta}[i]$ to $\boldsymbol{\theta}[i+1]$ corresponds with an observed object move from $\boldsymbol{\xi}[i]$ to $\boldsymbol{\xi}[i+1]$, this data can be used in the construction of learning samples which describe the actual behaviour of the camera-robot system.

Bins. The conjugate gradient method that is used to update the weights in the neural network minimises the summed squared error over a *set* of learning samples. Therefore the learning samples which are generated during control of the camera-robot system are collected in *bins*.

When n steps are used to move towards the object, $n \times (n-1)$ learning samples can be constructed. This can be seen from figure 6: for instance, apart from data obtained from the move $\boldsymbol{\theta}[0] \rightarrow \boldsymbol{\theta}[1]$ and $\boldsymbol{\theta}[1] \rightarrow \boldsymbol{\theta}[2]$, also the move $\boldsymbol{\theta}[0] \rightarrow \boldsymbol{\theta}[2]$ can be constructed by combining the previous two moves. With a typical value of n set to 100, a single trial may lead to nearly 10,000 samples. Clearly, the number of samples would grow quickly out of bound, leading to unacceptable delays in the learning algorithm, when all learning samples were kept.

Therefore a selective binning structure is set up as follows. Along each input dimension d of the neural network ($1 \leq d \leq 5$ in this case) a partition into $b[d]$ parts is made. This partitioning leads to an 5-dimensional structure of bins; a hypercube. When a learning sample is available, its input values uniquely determine which bin in the hypercube this learning sample fits in. When the corresponding bin is empty, the new sample is put in it; otherwise the sample in that bin is replaced by the new sample. Thus each bin contains only one learning sample.

An obvious advantage of the hypercube method to store the samples is that a uniform or otherwise desired distribution of the learning samples can be realised. This means that the neural network approximation will not lose accuracy in those parts of the input space which have not been visited for a long time.

Learning samples. The neural network $\mathcal{N}(\cdot)$ is trained with learning samples $(\xi, \theta; \Delta\theta)$. Unfortunately, it is not possible to analytically compute a $\Delta\theta$ from a given (ξ, θ), since that would require knowledge of the ideal controller $\mathcal{C}^0$. How can learning samples be constructed? In learning by interpolation the inputs to the neural controller do not only consist of $\theta[i]$ and $\xi[i]$ but also ξ_d. The network, which we will indicate by $\mathcal{N}'$, thus has eight inputs and three outputs. The task of the network is to generate a joint rotation

$$\mathcal{N}'\left(\xi[i], \theta[i], \xi_d\right) = \Delta\theta[i]$$

such that

$$\mathcal{R}\left(\xi[i], \theta[i], \Delta\theta[i]\right) = \left(\xi_d, \theta[i+1]\right).$$

The learning samples that are gathered, however, give information how the robot moves from $\xi[i]$ to $\xi[i+1]$, where $\xi[i+1]$ need not coincide with ξ_d. Thus the neural network learns the mapping

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

such that

$$\mathcal{R}\left(\xi[i], \theta[i], \Delta\theta[i]\right) = \left(\xi[i+1], \theta[i+1]\right).$$

This has the advantage that the neural controller can be used to move the robot to *any* target point, and that certain systematic errors (e.g., from the camera) are taken care of.

3.1.3 The control loop

In the first setup we are faced with the following system.

1. set $i = 1$, and set all measurements at $i = 0$ to zero. Set $\Delta\theta[1]$ to a small random value.

2. the robot control command (desired joint position) $\theta_d[i]$ is sent to the robot, and the robot moves.

3. the robot records the current joint values $\theta[i]$, as well as an image $\mathcal{I}$ from which the visual data $\xi[i]$ are extracted. The ξ consists of $\xi = (\xi_x, \xi_y, \xi_A)$ where

$$\begin{aligned}\xi_x &= x \text{ coordinate of target object in image}\\ \xi_y &= y \text{ coordinate of target object in image}\\ \xi_A &= \text{observed area (\# pixels) of target object in image.}\end{aligned}$$

4. we know that the robot moved from joint position $\theta[i-1]$ with joint rotation $\Delta\theta[i]$, and at the same time the visual data changed from $\xi[i-1]$ to $\xi[i]$. From this we can construct the following learning sample:

$$\mathcal{C}^0\big(\xi[i-1], \xi_d[i], \theta[i-1]\big) = \Delta\theta[i].$$

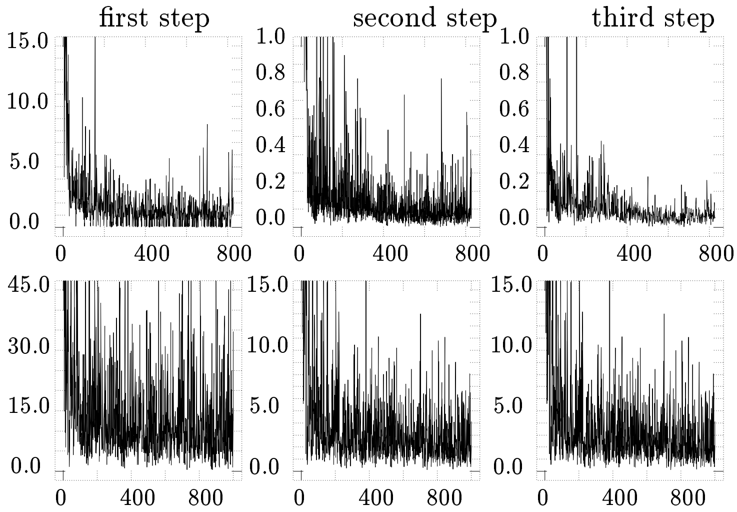

Figure 8: Grasping error in cm with the simulated robot after 1 (left column), 2 (second column), and 3 (third column) steps. On the horizontal axes, the number of goals is plotted. The desired accuracy was a Euclidean distance of 0.5cm between the end-effector and the target object. **Top row.** Initial learning: results when learning from scratch. **Bottom row.** After initial learning: rotation of the camera by 45°.

5. in the neural network are fed the visual and positional information $\xi[i]$ and $\theta[i]$, as well as the desired visual data at $i+1$, $\xi_d[i+1]$. In this case, θ is a *context variable* while ξ is the *control variable*. The network generates an output

$$\mathcal{N}'\big(\xi[i], \xi_d[i+1], \theta[i]\big) = \theta_d[i+1].$$

6. $i \leftarrow i+1$; goto 2.

3.1.4 Results

Results are shown in figure 8. The average error in the first step is less than 1cm and in two steps approximately 1mm. Note that after a few iterations, the target can already be reached in the feedback loop; such fast learning makes the method very well suited for real applications.

Secondly, the method's adaptability is tested. The state of the network after the initial 800 goals is taken as the begin state. The system is changed by rotating the hand, and hence the hand-held camera, by 45°. The result is shown in the bottom row of figure 8.

3.2 An experimental setup II

3.2.1 Introduction

The approach shown in the previous section has been demonstrated to be successful in practice. However, we made one simplification: it was assumed that the dimensions of the target object were known.

When the dimensions of the object are unknown, and vary from one run to another, this approach cannot be used. However, we can still use monocular vision, by use of *optic flow*.

Optic flow, which is defined as the motion of the observer's environment projected on its image plane, is in fact much more commonly used by living organisms than information obtained from static retinal patterns. The fact that optic flow is fundamental to vision has only been realised since the pioneering work of Gibson (Gibson, 1950). For instance, the gannet, when feeding itself, dives down in the water. During the dive, the bird needs its wings to adapt its path to the motion of the fish; however, at the moment of contact with the seawater its wings must be folded to prevent them from breaking. It has been shown that the time remaining between the moment that the bird folds its wings, and that it hits the water, is always the same for one particular bird. It is not controlled by its height or velocity separately, but the quotient of the two. This remaining time is known as the **time-to-contact** and indicated by the symbol τ. When the system is not accelerating, τ is given by the quotient of the distance from an approaching surface and the velocity towards it. Exactly the same information can be obtained from the divergence of the approaching surface (Koenderink & Doorn, 1975; Lee, 1980)—a feature that can be observed monocularly.

Figure 9: The gannet.

Since this bird cannot measure its velocity, it measures the time until it hits the water from time derivatives of the visual observation. It is this mechanism that underlies the method presented in this paper for controlling a monocular robot arm such that it 'lands' on an object. An ordinary grasping task can be described as: at some time, the distance between the object and the hand-held camera must be zero. We go one step beyond that requirement: the distance must be zero *at some time* (which, we now know, is related to the time-to-contact), while the system must be at rest: the velocity, acceleration, and higher derivatives must also be zero. We will call this the **goal state**. But this can be seen as the endpoint of a **trajectory** towards that point in time. In the case of the bird, the decision to fold its wings can be made with the available visually measured quantities. In the sequel it will be shown that by extending the above example to higher-order time derivatives of the visual data, criteria can be developed which specify a trajectory which ends in a rest state (i.e., zero velocity, acceleration, etc.) at the end point. These criteria will lead to **visual setpoints** along the followed trajectory, and are used as inputs to a neural controller which must generate robot joint accelerations in order to follow the setpoints in visual domain. Thus it is possible that the eye-in-hand robot arm exactly stops on an observed object by use of optic flow. By using time derivatives of the visual field we obtain to an important advantage: the model of the object is not needed to obtain depth information. The system need not be calibrated for each object that has to be grasped, but can approach objects with unknown dimensions.

3.2.2 Theoretical background

We can describe the motion of the camera—and, with it, the robot's end-effector—as a trajectory $\mathbf{x}_r$ in world space. This trajectory is known in terms of the robot variables $\boldsymbol{\theta}$; i.e., we know this trajectory in terms of a sequence of robot joint positions.

Secondly, we can describe the position of the target object as $\mathbf{x}_o$; this position is unknown and cannot be measured, since we have only a single camera and do not know the dimensions of the object. Recalling equation (4), we cannot even measure the position of the object *with respect to the camera*: if the object appears four times as small as before, it can be twice as far away, or four times as small, or any mixture of those two.

Let us define $\mathbf{d}(t) \equiv \mathbf{x}_r(t) - \mathbf{x}_o$; the unknown quantity $\mathbf{d}(t)$ is the *distance between* *the robot and object at time t.* The grasping task can be formulated as reaching the robot position $\boldsymbol{\theta}[\tau]$ at some desired time τ where $\mathbf{d}(t \geq \tau) = \mathbf{0}$. That is, *at some time τ and thereafter the position must be zero.* This can be realised when we require that $\mathbf{d}(\tau)$ as well as its derivatives are zero at τ; after all, when its derivatives are zero at that moment, $\mathbf{d}(\tau)$ will remain zero. The system will remain in rest. We call this the **stopping criterion**.

Assume that we describe each of the components of $\mathbf{d}(t)$ (i.e., the x, y, and z component as well as any rotational components) by a Taylor series,

$$d(t) = \sum_{j=0}^{n} a_i t^j + \epsilon. \tag{6}$$

The above stopping criterion can now be expressed as

$$\forall 0 \leq k < n : d^{(k)}(\tau) = \sum_{j=k}^{n} \frac{j!}{(j-k)!} a_j \tau^{j-k} = 0. \tag{7}$$

It can be shown (Smagt, 1995) that this leads to the following constraints on the parameters a_j:

$$\forall 0 \leq k < n : \frac{a_n^{n-k-1} a_k}{a_{n-1}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}. \tag{8}$$

Getting the visual data. Now, how are the a_j's related to the visual data that we can measure? Taking (4) into account again, we know that we can visually determine

$$\xi_x(t) = f\frac{d_x(t)}{d_z(t)}, \qquad \xi_y(t) = f\frac{d_y(t)}{d_z(t)}, \qquad \xi_A(t) = \frac{f^2 A}{d_z(t)^2}.$$

When we define

$$\xi'_x(t) \equiv \frac{d_x(t)}{\sqrt{A}}, \qquad \xi'_y(t) \equiv \frac{d_y(t)}{\sqrt{A}}, \qquad \xi'_z(t) \equiv \frac{d_z(t)}{f\sqrt{A}} \tag{9}$$

then again we can make Taylor polynomials for $\xi(t)$ such that

$$\xi(t) = \sum_{i=0}^{n} v_i t^i + o(t^n) \tag{10}$$

where $\xi(t)$ is either $\xi'_x(t)$, $\xi'_y(t)$, or $\xi'_z(t)$. Similarly, v_i indicates the x, y, or z parameters.

Once the parameters v_i are known, the polynomials for $\xi'_x(t)$, $\xi'_y(t)$, and $\xi_z(t)$ are known. Knowledge of these parameters, however, does not give sufficient information on the position, velocity, etc. of the end-effector relative to the object, since they are scaled by the constants A and f. However, the constraints can still be expressed in visual parameters: using the polynomial expansions for $d(t)$ and $\xi(t)$, and combining these with equations (9), the v_i's have a common constant

$$v_i = c a_i \tag{11}$$

where c is c_x, c_y, or c_z for the x, y, and z components of $\mathbf{d}$, given by

$$c_x = A^{-1/2}, \qquad c_y = A^{-1/2}, \qquad c_z = (f^2 A)^{-1/2}. \tag{12}$$

When we take (8) into account, we see that it can be rewritten as

$$\forall 0 \le k < n : \frac{{v_n}^{n-k-1} v_k}{{v_{n-1}}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}. \tag{13}$$

Since the equation contains the fraction $\frac{c^{n-1}}{c^{n-1}}$, the unknown constants disappear and we can determine (13) from visual measurements.

However, there are some problems with this equation. First, in a real-world system we would usually try to control second- or third-order trajectories; i.e., $n = 2$ or $n = 3$. This means that we have to visually determine v_0, v_1, v_2, and perhaps v_3. Knowing that the visual frame rate is low, and that visual data contains large amounts of noise, calculating v_2 is tough and v_3 is prohibitive.

Secondly, we cannot control the time that the trajectory lasts. This is determined by the initial conditions: for instance, if the robot initially moves fast and is not far away from the object, the trajectory will be quickly traversed. This poses a problem when the trajectory time differs for the x, y, and z components of the motion; it would be better if the robot arm moves in a straight path towards the object.

To solve both of these problems we introduce an extra constraint. Since $d(t)$ is approximated by a polynomial of order n, the n^{th} derivative of the approximation of $d(t)$ must be constant in time; a_n is constant. Therefore, the $n - 1^{\text{st}}$ must be linear in time. Consequently, the time to bring the $n - 1^{\text{st}}$ derivative to 0 is equal to the quotient of the two. For $n = 2$, this is the velocity divided by the acceleration. In the general case,

$$\tau = -\frac{a_{n-1}}{n a_n}. \tag{14}$$

This can be combined with the constraints (8), such that the system is now faced with n non-trivial constraints:

$$\frac{{a_n}^{n-k-1} a_k}{{a_{n-1}}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}, \quad 0 \le k < n, \quad \text{and} \quad \tau_d = -\frac{a_{n-1}}{n a_n}. \tag{15}$$

These constraints can be rewritten as

$$\frac{a_k}{a_{n-1}} = (-\tau_d)^{n-k-1} \frac{1}{n} \binom{n}{k}, \qquad 0 \le k \le n. \tag{16}$$

Similar to the time-independent case, satisfying these n non-trivial constraints leads to the desired trajectory. However, the constraints are all related and a simplification is in order.

Consider once again the polynomial expansion of $d(t)$ in (6). This polynomial expansion of order n is globally valid over the whole time interval, i.e., the whole trajectory of the robot arm. After splitting up the global time axis in intervals, the $d(t)$ can be *repeatedly* approximated in these intervals by polynomials with parameters $a_j[i]$. These approximations are written as

$$d[i](t[i]) = \sum_{j=0}^{n} a_j[i] t[i]^j + \varepsilon. \tag{17}$$

Note that $d(t) \equiv d[0](t[0])$, but that the parameters $a_j[i]$ are in general not equal to $a_j[i+1]$! The starting time t at which the $d[i]$ and thus the $a_j[i]$'s are defined is repeatedly changed.

As set out above, the task of the feed-forward based neural controller is to make the robot manipulator follow a pre-specified trajectory. During a time interval i the system measures the $\boldsymbol{\xi}[i]$ (note that, due to the discrete-step feedback loop, the $\boldsymbol{\xi}$ are now discrete variables indexed by the step number instead of varying continuously in time). From these measurements the controller generates joint accelerations $\ddot{\theta}[i+1]$ and sends them to the robot. This marks the beginning of a new time interval $i+1$.

Now, by using the repeatedly updated $d[i]$, we can combine the constraints (15) and find the simplified form

$$\frac{a_0[i]}{a_1[i]} = -\frac{(\tau_d - t[i])}{n}, \qquad \forall i : 0 \leq i < \nu, \tag{18}$$

where $\nu \geq n$ and $(\tau_d - t[\nu]) \geq 0$. The proof of this theorem can be found in (Smagt, 1995). We will refer to (18) as the **time-dependent constraint**.

The time-dependent constraint is obtained by extending the *local* time intervals towards the moment when $(\tau_d - t[i]) = 0$. Although the $d[i](t[i])$ is a local approximation, we can just pretend that it fits the *whole* $d(t)$ starting at $t[i] = 0$, and let the time-dependent constraint be valid for that trajectory.

τ_d Is usually chosen equal for the x, y, and z components of $\mathbf{d}(t[i])$, to ensure that all components go to zero at the same time.

3.2.3 The control loop

The time-dependent constraint (18) is used in a controller as follows.

1. set $i = 1$, and set all measurements at $i = 0$ to zero. Set $\ddot{\theta}_d[1]$ to a small random value.

2. the robot control command (desired joint acceleration) $\ddot{\theta}_d[i]$ is sent to the robot, and the robot moves.

3. the robot records the current joint values $\theta[i]$ from which the $\dot{\theta}[i]$ and $\ddot{\theta}[i]$ are computed, as well as an image $\mathcal{I}$ from which the visual data $\boldsymbol{\xi}[i]$ are extracted. The $\boldsymbol{\xi}$ consists of $\boldsymbol{\xi} = (\xi_x, \dot{\xi}_x, \xi_y, \dot{\xi}_y, \xi_A, \dot{\xi}_A)$ where

$$\begin{aligned} \xi_x &= x \text{ coordinate of target object in image} \\ \xi_y &= y \text{ coordinate of target object in image} \\ \xi_A &= \text{observed area (\# pixels) of target object in image.} \end{aligned}$$

4. Set $\mathbf{r}[i] = a_0[i]/a_1[i] - (\tau_d - t[i])/n$ for the x, y, and z directions.

   we know that the robot moved from joint position $\theta[i-1]$ with joint rotation $\theta_d[i]$, and at the same time the visual data changed from $\boldsymbol{\xi}[i-1]$ to $\boldsymbol{\xi}[i]$. From this we can construct the following learning sample:

$$\mathcal{C}^0\big(\mathbf{r}[i-1], \mathbf{r}[i], \theta[i-1], \dot{\theta}[i-1], \ddot{\theta}[i-1]\big) = \ddot{\theta}_d[i].$$

5. Apply the newly measured data to the network. The network generates an output

$$\mathcal{N}\big(\mathbf{r}[i], 0, \theta[i], \dot{\theta}[i], \ddot{\theta}[i], \big) = \ddot{\theta}_d[i+1].$$

6. $i \leftarrow i + 1$; goto 2.

The control loop is depicted in figure 10.

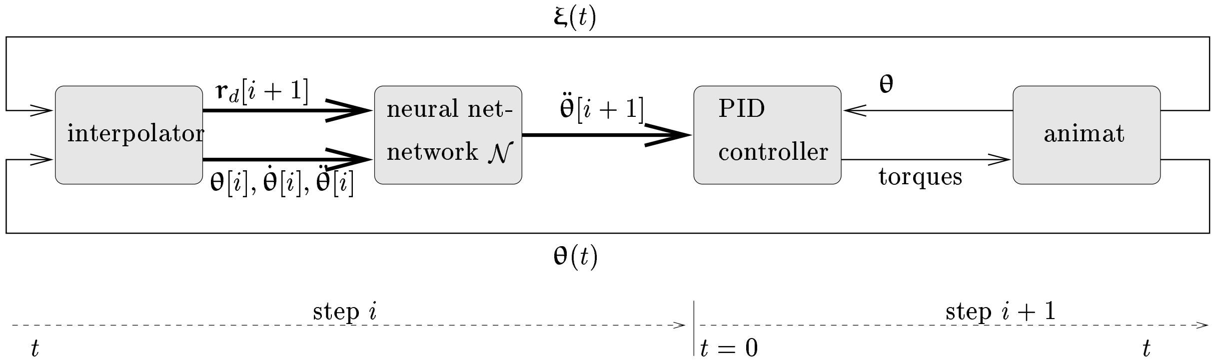

Figure 10: The structure of the time-to-contact control loop.

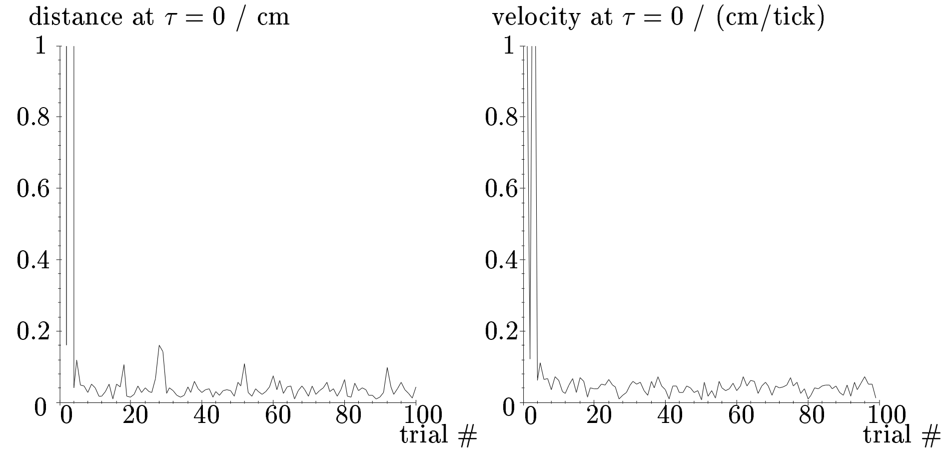

Figure 11: Distance and velocity at $\tau = 0$. The left figure shows the distance $\sqrt{d(\tau=0)_x^2 + d(\tau=0)_y^2 + d(\tau=0)_z^2}$ between the end-effector and the approached object. The right figure shows the velocity $\sqrt{\dot{d}(\tau=0)_x^2 + \dot{d}(\tau=0)_y^2 + \dot{d}(\tau=0)_z^2}$ of the end-effector in cm/tick. Typical end-effector velocities during the trajectory are between 0.5 and 2.0. The horizontal axis indicates the trial number.

3.2.4 Results

In order to measure the success of the method while applied to the simulated robot, we measure the $\mathbf{d}(t)$, $\dot{\mathbf{d}}(t)$, and $\ddot{\mathbf{d}}(t)$ during the trajectory; with the simulator, these data are available. A correct deceleration leads to a $\mathbf{d}(t) = \dot{\mathbf{d}}(t) = 0$ when $\tau_d = 0$, i.e., at the end of the trajectory. The results of a run with the simulated OSCAR robot are shown in figure 11. This graph shows the distance between the end-effector and the target object at $\tau_d = 0$. The results show that after only a few trials (in this case, 4), the positional error at $(\tau_d - t[0]) = 0$ is below one millimeter, while the velocity is below 0.1cm per simulator time unit (typical end-effector velocities during deceleration are 0.5–2.0 cm per simulator time unit).

Figure 12 shows the results of the control algorithm when tested with different noise levels. Noise is present in the measurements, and also in the learning samples which are taught to the neural network. A graceful degradation is shown up to very high noise levels, as high as 7 times the expected amount of noise.

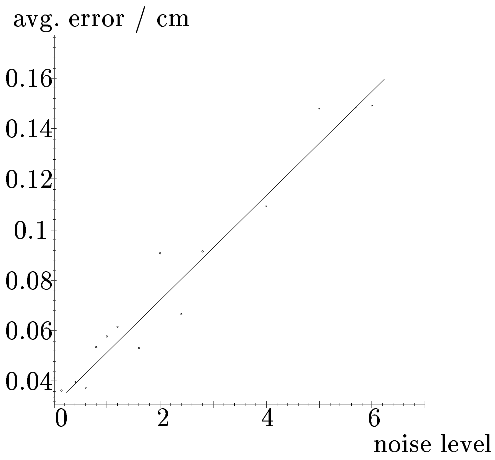

Figure 12: Results of the simulation when noise in the visual input is applied. The figure shows the distance $\sqrt{d(\tau=0)_x^2 + d(\tau=0)_y^2 + d(\tau=0)_z^2}$ between the end-effector and the approached object averaged over 200 goals (marked by dots in the graph). The vertical axis shows the noise level l. The figure clearly shows that the noise level and the grasping error are linearly related. At values of $l \geq 7$ the signal-to-noise ratio is thus large that the system cannot always reach the target at $\tau_d = 0$, but sometimes overshoots. The average error at $l = 7$ goes up to 2.0cm; at $l = 8$ it is as high as 4.0 due to overshoots.

Acknowledgments

Continuing support by Gerd Hirzinger, who leads the Department of Robotic Systems at the German Aerospace Research Establishment is kindly acknowledged. The author is grateful for the many fruitful discussions with Frans Groen which were essential for this work.

References

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

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

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

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

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.* New York: McGraw-Hill Book Company.

Gavrila, D. M., & Groen, F. C. A. (1992). 3D object recognition from 2D images using geometric hashing. *Pattern Recognition Letters*, *13*, 263–278.

Gibson, J. J. (1950). *The perception of the visual world.* Houghton Mifflin.

Hecht, E., & Zajac, A. (1974). *Optics.* Addison-Wesley Publishing Company.

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

Koenderink, J. J., & Doorn, A. J. van. (1975). Local structure of the motion parallax. *Optica Acta*, *22*(9).

Lamdan, Y., & Wolfson, H. J. (1988). Geometric hashing: A general and efficient model-based recognition scheme. In *Proceedings of the Second International Conference on Computer Vision.* New York: IEEE.

Lee, D. N. (1980). The optic flow field: the foundation of vision. *Phil. Trans. R. Soc. Lond. B*, *290*, 169–179.

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.

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

Smagt, P. van der. (1994). Minimisation methods for training feed-forward networks. *Neural Networks*, *7*(1), 1–11.

Smagt, P. van der. (1995). *Visual Robot Arm Guidance using Neural Networks.* Unpublished doctoral dissertation, Dept of Computer Systems, University of Amsterdam.

Smagt, P. van der, Jansen, A., & Groen, F. (1992). Interpolative robot control with the nested network approach. In *IEEE Int. Symposium on Intelligent Control* (p. 475-480). Glasgow, Scotland, U.K.: IEEE.