3

In: **Neural Systems for Robotics**
O. Omidvar and P. van der Smagt, editors
pp. 37–73
Academic Press, 1997

Visual Feedback in Motion

Patrick van der Smagt
Frans Groen

ABSTRACT In this chapter we introduce a method for model-free monocular visual guidance of a robot arm. The robot arm, with a single camera in its end effector, should be positioned above a stationary target. It is shown that a trajectory can be planned in visual space by using components of the optic flow, and this trajectory can be translated to joint torques by a self-learning neural network. No model of the robot, camera, or environment is used. The method reaches a high grasping accuracy after only a few trials.

3.1 Introduction

Living organisms, in contrast with static robot control systems, use continuous feedback from their eyes in order to interact with their dynamically changing environment. In the meantime, there is sensory activity due to self-motion which is taken care of. While the information that is used in a static system only takes into account the *position* of points of interest, in a moving system such visual observations are interpreted in spatio-temporal domain. *Optic flow*, which is defined as the motion of the observer's environment projected on its image plane, is 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 realized since the pioneering work of Gibson [3].

A classic example of the use of optic flow in biological systems is the

FIGURE 3.1. The gannet.

gannet,[1] shown in Figure 3.1. When hunting for fish, this bird drops itself from the sky from a great height. Since the fish is moving, the bird needs its wings to correct its path while falling; 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 between the moment the bird folds its wings and when it hits the water is always the same for one particular bird. It is not controlled by its height or velocity separately, but by 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 toward it. Exactly the same information can be obtained from the divergence of the approaching surface [4,5]—a feature that can be observed monocularly.[2]

Since this bird cannot measure its velocity, it measures the time until it hits the water from the time derivatives of visual observation. It is this mechanism that underlies the method presented in this chapter for controlling a monocular robot arm such that it "lands" on an object. In one aspect, the task that is tackled can be simply described as follows: 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

[1]The gannet is a seabird from the Sulidæ family (related to the pelican) and is common in the northern parts of the Atlantic Ocean, as well as in more temperate regions (Africa, Australasia, and South America). The bird, which has a span of 1.8 m, has white feathers except for its black primary wing feathers, and its yellowish neck and head equipped with a long, straight beak. In the brooding season it nests in large colonies of up to 60,000 birds on cliffs and islands. It feeds itself by diving for fish from a great height, and following the fish under water.

[2]Considering the fact that birds have very small binocular fields [6], it is indeed very unlikely that binocular vision can be used for such a task.

must also be zero. We will call this the *goal state*. But this can be seen as the end point of a *trajectory* toward 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 the visual domain. Thus it is possible that the eye-in-hand robot arm stops exactly on an observed object by use of optic flow. Using time derivatives of the visual field leads 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.

The use of optic flow for robot navigation is no novelty. Cipolla and Blake [2], for example, describe a system for estimating time to contact from the flow field, and use this for obstacle avoidance of a robot arm with known kinematics. Sharma [7] uses time to contact derived directly from the optic flow as a basis for mobile robot trajectory planning. Vernon and Tistarelli [8] use visual velocity vectors to estimate depth in a complex visual scene, but again for a robot with known kinematics. We generalize previous time to contact-based robot arm guidance methods to generation of three-dimensional motion trajectories from optic flow, and use this to construct a model-free self-learning robot arm controller (in fact, not the optic flow field will be used but rather the visual position, velocity, acceleration, etc., of a single white object against a black background). Of importance is the fact that no model of the robot arm or the observed world is necessary to obtain depth information with monocular vision.

First, we will investigate which visual criteria are necessary to let the robot arm follow the desired trajectory toward an object. Second, it will be shown how these criteria can be used as reference signals for a self-learning neural controller. This controller will use no model of the robot arm or camera, and will also not need a model of the approached object. In Section 3.2, the trajectory followed by the hand-held camera point will be mathematically investigated. This leads to the time-independent motion constraints in Section 3.3, and the time-dependent motion constraints in Section 3.4. Next, the question of how such motion can be observed with a camera moving in its environment will be studied in Section 3.5. The conclusions of the preceding sections lead to a control algorithm, described in Section 3.6. Results of the algorithm applied to a robot are given in Section 3.7. A discussion in Section 3.8 concludes the chapter.

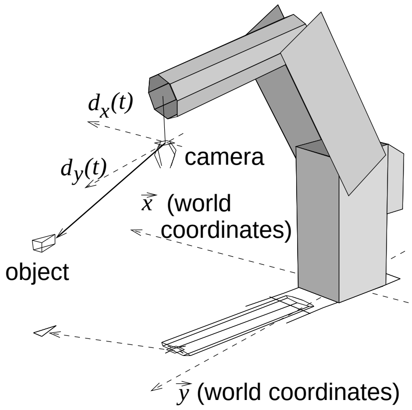

FIGURE 3.2. Setup of OSCAR trying to grasp an object. The object is observed by the camera at position $\mathbf{d}(t=0)$, relative to the coordinate system of the end effector. Subsequent measurements of the object occur at subsequent time steps. The figure also shows the projections of the vectors and objects on the $z = 0$ plane.

3.2 The Trajectory of an Eye-in-Hand System

A typical configuration of the object relative to the manipulator is shown in Figure 3.2. The task of the neural controller is to generate robot joint accelerations $\ddot{\boldsymbol{\theta}}$ which make the robot manipulator follow a trajectory leading to the goal state in which the end effector (c.q., the camera) is placed on the object, and the robot arm is at rest. This trajectory will be expressed *in the available visual domain*. Therefore, visual criteria will be determined to which the visually measured trajectory of the observed object will have to adhere. These criteria have to be defined from the visual observations $\boldsymbol{\xi}(t)$, and we can write the goal state as

$$\forall t \geq \tau : \quad \boldsymbol{\xi}(t) = \boldsymbol{\xi}_d. \tag{3.1}$$

Note that the discrete iteration step i is replaced by the continuous t, and the step number n_τ by τ, the time to contact.

However, for reasons of clarity we will initially describe that trajectory as well as the goal state in world space, which will lead to constraints on the parameters of that trajectory. As will be shown subsequently, these constraints then can also be used in visual space.

Thus the moving eye-in-hand robot arm and its environment will be regarded as two points: a static object and a moving camera. The world positions of these points are given by $\mathbf{x}_o$ and $\mathbf{x}_r(t)$, respectively. Furthermore, we introduce the distance $\mathbf{d}(t)$ in world space as follows.

Definition 3.1 *The distance between the camera and object is denoted by the symbol* $\mathbf{d}(t)$*, and is determined from the Cartesian positions* $\mathbf{x}_t(t)$ *of* *the camera and* $\mathbf{x}_o$ *of the object as*

$$\mathbf{d}(t) \equiv \mathbf{x}_r(t) - \mathbf{x}_o. \tag{3.2}$$

The *distance* of the observer relative to the object is described by $\mathbf{d}(t)$. We will use $\mathbf{d}(t)$ to determine the trajectory that must be followed in the x, y, and z directions by the manipulator to reach the target. Note that, with a single camera, $\mathbf{d}(t)$ cannot actually be measured without model knowledge of the observed object, but the resulting constraints on the trajectory will be expressible in the visual domain.

Equation (3.1) itself does not lead to three-dimensional criteria to determine when the target is reached without using multiple cameras or a model of the observed object. Since we want to use neither, we have to find another criterion based on the relative Cartesian position of the object $\mathbf{d}(t)$.

Definition 3.2 *The stopping criterion is equivalent to reaching the state where the relative distance is zero at all times* $t > \tau$*:*

$$\forall t \geq \tau : \quad \mathbf{d}(t) = 0. \tag{3.3}$$

In other words, the system is at rest. Equation (3.3) is regarded as the most important stopping criterion that is set to the trajectory and is followed by the camera before it reaches the object. Equation (3.3) can also be interpreted as: All time derivatives of $\mathbf{d}(t)$ must be zero at $t = \tau$.

Corollary 3.1 *The stopping criterion of Definition 3.2 can be expressed as*

$$\forall k \geq 0 : \quad \mathbf{d}^{(k)}(\tau) = 0. \tag{3.4}$$

In other words, the distance, velocity, and higher derivatives must become zero at the moment of grasping.

We will now derive how the stopping criterion (3.4) can be used to determine the trajectory $\mathbf{d}(t)$ *before* the time of contact (i.e., when $t < \tau$).

3.2.1 The Trajectory as a Function of Time

In order to answer this question we will first partition $\mathbf{d}(t)$ into its x, y, and z components such that $\mathbf{d}(t) = (d_x(t), d_y(t), d_z(t))$, as illustrated in Figure 3.2. In the sequel, the symbol $d(t)$ will be understood to mean any of $d_x(t)$, $d_y(t)$, or $d_z(t)$.

As the trajectory is described in terms of its time derivatives, and these time derivatives describe the physical concepts of velocity, acceleration, etc., a natural choice is to approximate $d(t)$ by a finite-order polynomial expansion around $t = 0$:

$$d(t) = \sum_{j=0}^{n} a_j t^j + \varepsilon, \tag{3.5}$$

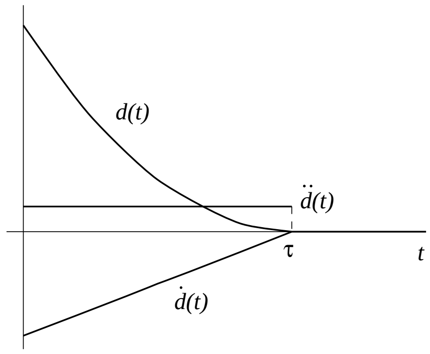

FIGURE 3.3. A possible trajectory followed by the manipulator. In addition to $d(t)$, the first and second derivatives of $d(t)$ are also shown. At $t = \tau$, a discontinuity is applied to $\ddot{d}(t)$: The deceleration is "switched off." The relative scale of the three curves is distorted. Although the sign of $\ddot{d}(t)$ is positive, we will indicate it as "deceleration" due to the fact that its sign is opposite that of $\dot{d}(t)$.

where $a_n \neq 0$ and ε is the approximation error, which equals $o(t^n)$ for small t. Here, the nth derivative is the highest derivative of $d(t)$ that is taken into consideration. Therefore, also, the stopping criterion (3.4) will only be taken into consideration for $0 \leq k < n$. The derivatives of $d(t)$ are given by

$$d^{(k)}(t) = \sum_{j=k}^{n} \frac{j!}{(j-k)!} a_j t^{j-k} + o(t^n).$$

Differently put, the derivatives of $d(t)$ and the a_k are related as

$$a_k = \frac{1}{k!} d^{(k)}(t = 0). \tag{3.6}$$

What does a trajectory $d(t)$ look like? Consider Figure 3.3, which shows a system where $d(t)$ is taken to be a quadratic trajectory in time, i.e., where $n = 2$ leading to a trajectory with constant deceleration. Note that at $t = \tau$ the criterion (3.4) has indeed been satisfied for $0 \leq k < 2$. The system has reached the target and is at rest, as required. In order to prevent the system from moving away from 0 again, the deceleration is externally "switched off" at $t = \tau$ as shown. The definition of $d(t)$ as a polynomial is related to its physical interpretation: We consider the trajectory followed by the manipulator in terms of position $d(t)$, velocity $\dot{d}(t)$, deceleration $\ddot{d}(t)$, etc., with respect to time. Thus the $n = 2$ case is a physically familiar one: gravity-induced motion, or, more general, motion resulting from a constant force. The $n = 3$ case, where the jerk (the time derivative of deceleration) is constant, will also be considered: Large deceleration differences cannot always be realized by mechanical structures (robot arms), thus asking for a more smooth motion profile. In the $n = 3$ case, the system is thus controlled so that the deceleration also goes to 0 at $t = \tau$ and the jerk is switched off at $t = \tau$.

We proceed with substituting the $d(t)$ and its derivatives in criterion (3.4). This criterion will lead to constraints on the parameters of the trajectory, which can be used by the adaptive controller.

3.3 The Time-Independent Constraints

The parameters a_k uniquely determine the trajectory $d(t)$ that is followed by the manipulator in the x, y, and z direction of motion. The stopping criterion (3.4) therefore leads to conditions on these parameters a_k. Thus the conditions on the trajectory at the *end* are translated to conditions *along* the trajectory.

Although the conditions on the parameters a_k can be expressed in various ways, since they are used as input to the controller it is imperative that they be measurable from the visual data. In Section 3.5 it will be derived that, although the parameters a_k cannot be measured, they *are* known scaled by an unknown constant. Therefore, only the relative quantities a_j/a_k are known, since only in those expressions do the constants of proportionality disappear. Therefore we have to find criteria in which the relative quantities a_j/a_k appear.

In the previous section we formulated the requirements for the trajectory that is followed by the manipulator in time. These requirements can be summed up as follows:

1. $d(t \geq \tau) = 0$.
2. $d(t)$ can be well approximated by a polynomial of finite order.

The criteria for the a_i to meet these conditions can be found from the stopping criterion (3.4), which requires that $d(\tau) = \dot{d}(\tau) = \ddot{d}(\tau) = \ldots = 0$. This results in the following theorem.

Theorem 3.1 *Each of the components $d(t)$ of $\mathbf{d}(t)$ is described by a different polynomial*

$$d(t) = a_0 + a_1 t + \ldots + a_n t^n.$$

Then the stopping criterion

$$\exists \tau \; \forall k, 0 \leq k < n : \quad d^{(k)}(\tau) = 0,$$

where $\tau > 0$ is an unspecified moment in time, leads to the following set of simultaneous constraints on the parameters of the trajectory:

$$\frac{{a_n}^{n-k-1} a_k}{a_{n-1}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}, \qquad 0 \leq k < n. \tag{3.7}$$

Proof: The proof is given in Appendix 3.A. ■

The $k = n - 1$ case reduces to a trivial equation; therefore, Eq. (3.7) leads to $n - 1$ nontrivial constraints. Since τ is not a parameter of these constraints, we will refer to (3.7) as the *time-independent constraints.*

The problem that is tackled can be formulated as: Given a moving system $d(t) = \sum_{i=0}^{n} a_i t^i$ which is in state $a_0, a_1, \ldots, a_{n-1}$, find the a_n such that at some time τ, $d(\tau) = \dot{d}(\tau) = \cdots = d(\tau)^{(n-1)} = 0$. Formulated as this, we see that there are n equations with two unknowns (τ and a_n). Therefore, a solution can only be guaranteed when $n = 2$. In practice this means that, given a system at position $d(t)$ moving with velocity $\dot{d}(t)$, you can always find a constant deceleration $\ddot{d}$ such that $d(t) = \dot{d}(t) = 0$ at some time. Cases for $n > 2$ are, in general, not soluble. For instance, considering the $n = 3$ case, it is not always possible to find a linearly decreasing deceleration with which the velocity and position also go to 0 at some time $t = \tau$.

To clarify these statements, we will consider the case $n = 2$ in more depth, and illustrate the impossibility of the $n = 3$ case for a constant jerk a_3 along the whole trajectory.

Quadratic Polynomials

For $n = 2$, (3.7) reduces to the single equation

$$n = 2: \qquad \frac{a_2 a_0}{{a_1}^2} = \frac{1}{4}. \tag{3.8}$$

Lemma 3.1 *(model equation) The time-independent constraints for the $n = 2$ case lead to an optimal deceleration*

$$a_2 = \frac{{a_1}^2}{4a_0}. \tag{3.9}$$

Proof: The proof follows immediately from (3.7) when τ is finite. ■

Equation (3.9) gives the required deceleration to lead the system to the state where the position and velocity are zero at the same time. Thus the condition (3.4) is satisfied for $k < 2$.

Equation (3.9), when written as $a_2 a_0 / {a_1}^2 = 1/4$, is a constraint on the measurable relative quantities a_2/a_1 and a_0/a_1. It is therefore possible to design a controller which uses these measurable quantities for a system to perfectly decelerate toward an object, by having it satisfy the constraint on those quantities. Note that the time until contact τ cannot be controlled; it is, in fact, equal to $2a_0/a_1$ and thus depends on the initial position and velocity of the system only. This case has been investigated by Lee [5].

Cubic Polynomials

As mentioned above, for $n > 2$, no solutions exist in the general case. As an example, for $n = 3$, (3.7) reduces to the combined equations

$$n = 3: \qquad \frac{{a_3}^2 a_0}{{a_2}^3} = \frac{1}{27}, \quad \frac{a_3 a_1}{{a_2}^2} = \frac{1}{3}. \tag{3.10}$$

This set of equations leads to two, possibly different, values for a_3 which solve the equations. In order to bring the system to the desired rest state as given in (3.4), where the position, velocity, and deceleration are zero, both solutions must give the same result; otherwise, the desired rest state cannot be reached. The solution to (3.10) describe a line in the three-dimensional space spanned by a_0, a_1, and a_2. Clearly, there exist initial values of a_0, a_1, and a_2 which do not lie on this line; thus, for those initial values a constant a_3 does not exist to follow that line. This problem can be solved by taking a *two-step* approach, where a different a_3 is given in the first and second steps. In the first the system will be brought on the desired (a_0, a_1, a_2) line, and in the second step this line will be followed. The following section investigates this case further.

Discussion

There are a few problems with the above solution. First, although, in the $n = 2$ case the optimal deceleration can be found to bring the system to the desired state, where the position and velocity are zero, the *time* over which this deceleration is performed cannot be controlled. This is clear from (3.7): τ is not a parameter of the system. Therefore, the time needed for deceleration is solely dependent on the starting conditions, e.g., the position and velocity in a second-order system at the moment that deceleration is initiated. Figure 3.4 illustrates this effect. It means that, with a system controlling quadratic trajectories, it may happen that the distance $d(t)$ is large while the velocity $\dot{d}(t)$ is small, such that it will take a very long time before the system will reach the steady state. A result of this problem is that, if τ cannot be controlled, it will in general be different for $d_x(t)$, $d_y(t)$, and $d_z(t)$. In other words, although the form of the one-dimensional trajectory is controlled, the trajectory followed in two- or three-dimensional Cartesian space *cannot* be controlled; it need not be straight. One such possible trajectory for the two-dimensional case is depicted in Figure 3.5. The figure illustrates that, while the y component of $\mathbf{d}(t)$ is still nonzero and decreasing, the x component is already 0. A second problem is that the above solution only works for $n = 2$. To solve these problems, the stopping criteria have to be expanded to incorporate τ.

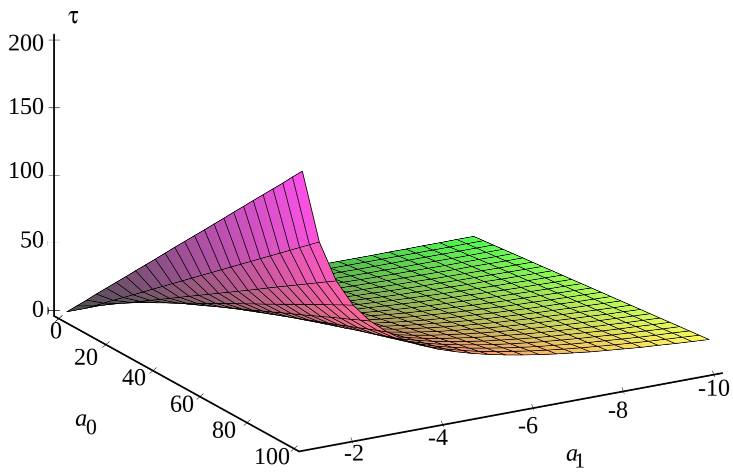

FIGURE 3.4. Time to contact depends on beginning conditions. The τ depends on the values of a_0 (position) and a_1 (velocity) along the trajectory.

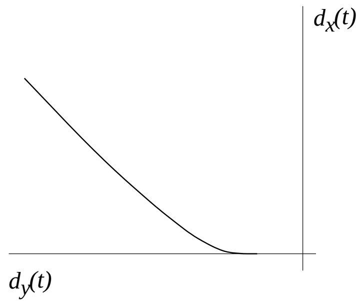

FIGURE 3.5. A planar trajectory followed when the x and y trajectories are not synchronized. Both the x and y trajectories that lead to this figure are quadratic polynomials. Note that the x coordinate will become 0 before the y coordinate does.

3.4 The Time-Dependent Constraints

The solution found in the previous section, where a constant deceleration is applied, leads to a minimum-time path. In this case we used a *single-step approach*: During the whole deceleration, the a_2 is kept at a constant value. The advantage of this method is that it results in a minimum-time path when the maximal allowed acceleration and deceleration is chosen. This method, however, has a disadvantage: The large acceleration differences are difficult to realize by mechanical structures, leading to increased wear and tear. Second, as we saw in the previous section, it is imperative that the *desired* time to contact τ_d be a parameter of the system.

A *multistep approach*, in which the deceleration a_2 is not constant along the whole trajectory, must be used to tackle these two problems. Since the deceleration is no longer constant along the trajectory, it must be repeatedly adapted, which can be realized by feedback control. Here the deceleration is assumed constant in each time interval, but varies from interval to interval.

3.4.1 Feedback Control of the Trajectories

We consider time in *intervals* i, $i+1$, ... In each interval i is defined a time axis $t[i]$ with corresponding trajectory parameters $a_k[i]$. Note that the intervals need not be equidistant.

Consider once again the polynomial expansion of $d(t)$ in (3.5). 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 into intervals, the $d(t)$ can be *repeatedly* approximated in these intervals by polynomials with parameters $a_j[i]$. These approximations are written as

$$d[i](t[i]) = \sum_{j=0}^{n} a_j[i]t[i]^j + \varepsilon. \tag{3.11}$$

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 feedforward-based neural controller is to make the robot manipulator follow a prespecified 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 $\ddot{\boldsymbol{\theta}}[i+1]$ and sends them to the robot. This marks the beginning of a new time interval $i+1$.

In Figure 3.6 a trajectory is shown which is realized by feedback control. In this figure, the deceleration is computed for each time interval i, $i+1$, ..., leading to a continuously adapted $d(t)$ and $\dot{d}(t)$.

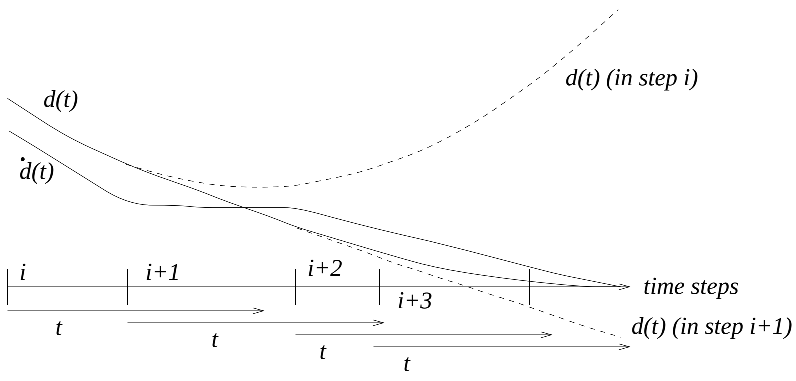

FIGURE 3.6. Exemplar trajectory followed by a feedback-controlled manipulator in a time scale. Shown are the position $d(t)$ and velocity $\dot{d}(t)$ during the decelerated motion (both are scaled to fit in the figure, and their signs are not shown). The figure starts at i, when the first deceleration is applied. From i to $i+1$, new measurements lead to a newly estimated deceleration, and thus the motion profile changes at $i+1$. Ditto at $i+2$, from when on the motion profile of $i+2$ is supported in the subsequent time slices. The dashed lines give the predicted $d(t)$ from i and $i+1$, which would have been followed if the deceleration would not have been changed.

To find the relationship between $a_k[i]$ and $a_k[i+1]$, we equate $d[i](t[i])$ and $d[i+1](t[i+1]-\Delta t[i])$, where $\Delta t[i]$ equals the time difference between the $t=0$ for $d[i]$ and $d[i+1]$:

$$\begin{aligned}
a_0[i+1] &= a_0[i] + a_1[i]\,(t[i]-t[i+1]) + \ldots + a_n[i]\,(t[i]-t[i+1])^n\,, \\
a_1[i+1] &= a_1[i] + a_2[i]\,(t[i]-t[i+1]) + \ldots + na_n[i]\,(t[i]-t[i+1])^{n-1}\,, \\
&\cdots \\
a_k[i+1] &= \sum_{j=k}^{n} \binom{j}{k} a_j[i]\,(t[i]-t[i+1])^{j-k}\,, \qquad (3.12) \\
&\cdots \\
a_n[i+1] &= a_{n-1}[i] + na_n[i].
\end{aligned}$$

From these equations we can see a clear correspondence between the $a_k[i]$'s and the $d[i]^{(k)}$'s:

$$d[i]^{(k)}(0) = k!\,a_k[i], \qquad (3.13)$$

in correspondence with (3.6) and (3.11).

3.4.2 The Time-Dependent Constraints

We will expand the criterion (3.7) by introducing a desired time to contact τ_d. The conditions that now lead to the trajectory are:

1. $d(t \geq \tau_d) = 0$.

2. $d(t)$ can be well approximated by a polynomial of finite order.

This can be done as follows. Since $d(t)$ is approximated by a polynomial of order n, the nth derivative of the approximation of $d(t)$ must be constant in time; a_n is constant. Therefore, the $n-1$st must be linear in time. Consequently, the time to bring the $n-1$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 (cf. Appendix 3.A),

$$\tau = -\frac{a_{n-1}}{n a_n}. \tag{3.14}$$

This can be combined with the constraint (3.7) such that the system is now faced with n nontrivial constraints:

$$\frac{{a_n}^{n-k-1} a_k}{a_{n-1}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}, \quad 0 \leq k < n, \quad \text{and} \quad \tau_d = -\frac{a_{n-1}}{n a_n}. \tag{3.15}$$

These constraints can be rewritten as

$$\frac{a_k}{a_{n-1}} = (-\tau_d)^{n-k-1} \frac{1}{n} \binom{n}{k}, \qquad 0 \leq k \leq n. \tag{3.16}$$

Note that, again, a trivial case results for $k = n - 1$. Similar to the time-independent case, satisfying these n nontrivial constraints leads to the desired trajectory. However, the constraints are all related and a simplification is in order. This leads to the following theorem.

Theorem 3.2 *Each of the components $d(t)$ of $\mathbf{d}(t)$ is described by a polynomial*

$$d(t) = a_0 + a_1 t + \ldots + a_n t^n.$$

Then the stopping criterion

$$\forall k, 0 \leq k < n: \quad d^{(k)}(\tau_d) = 0,$$

where τ_d is the desired time to contact, leads to the following constraint on the parameters of the successive intervals i of the trajectory:

$$\frac{a_0[i]}{a_1[i]} = -\frac{(\tau_d - t[i])}{n}, \qquad \forall i : 0 \leq i < \nu, \tag{3.17}$$

where $\nu \geq n$ and $(\tau_d - t[\nu]) \geq 0$.

Proof: The proof is given in Appendix 3.B. ∎

We will refer to (3.17) as the *time-dependent constraint.*

The time-dependent constraint is obtained by extending the *local* time intervals toward 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* $d(t)$

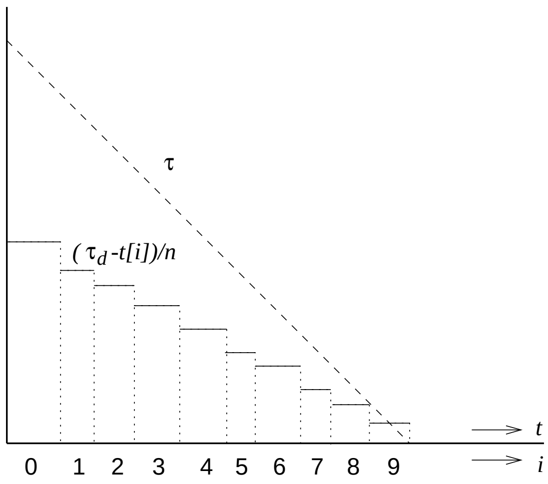

FIGURE 3.7. Desired time to contact and resulting constraint. Whereas τ_d decreases linearly in time, the constraint $(\tau_d - t[i])/n$ (shown for $n = 2$) changes only between interval i and $i+1$. Note that the intervals need not be equidistant in time.

starting at $t[i] = 0$, and let the time-dependent constraint be valid for that trajectory. This is clearly illustrated in Figure 3.6; in this case, however, the constraint is repeatedly applied to each local approximation to $d[i](t[i])$. Note that, since the time-dependent constraint is valid for each beginning point of a local approximation to $d(t)$, it is valid for each $t[i]$.

The value of 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.

Discussion

Theorem 3.2 shows that the trajectory of any n-order system can be described by a single constraint on the ratio of position and velocity of that system. The n *simultaneous* criteria (3.16) which have to be satisfied at some time are replaced by satisfying a *single* constraint (3.17) during n subsequent steps. The "simultaneous conditions" are "laid out in time." Equation (3.17) leads to a true feedback system: Not only is the feedback loop necessary to manage the imperfect control system, it is inherent to the control law.

Note the important advantage with respect to the time-independent constraints: The trajectory can be expressed by only the setpoint $(\tau_d - t[i])$ and $a_0[i]$ and $a_1[i]$; higher-order $a_k[i]$'s are not required. Figure 3.7 shows an example of how the setpoints $(\tau_d - t[i])/n$ vary from time interval i to $i+1$ for the case $n = 2$, as the desired time to contact τ_d goes to 0.

As before, we formulate the problem that is tackled as: Given a moving system $d(t) = \sum_{i=0}^{n} a_i t^i$ which is in state a_0, a_1, …, a_{n-1}, find the a_n

such that at *the given* time τ, $d(\tau) = \dot{d}(\tau) = \ldots = d(\tau)^{(n-1)} = 0$. At each step there is one equation (3.17) that has to be satisfied, and to realize this deceleration $a_2[i]$ in the time interval i can always be found.

Smoothness of the Deceleration Path

As an illustration, let us assume quadratic polynomial trajectories in the x, y, and z directions of motion.

Lemma 3.2 *(model equation) The time-dependent constraints for quadratic polynomials lead to an optimal deceleration*

$$a_2[i] = -\frac{na_0[i] + a_1[i](n + (\tau_d - t[i]) - 1)}{n + 2(\tau_d - t[i]) - 2}. \tag{3.18}$$

Proof: When taking (3.12) for $n = 2$, i.e.,

$$a_0[i+1] = a_0[i] + a_1[i] + a_2[i], \tag{3.19}$$

$$a_1[i+1] = a_1[i] + 2a_2[i] \tag{3.20}$$

and substituting a_0 and a_1 in (3.17) for $i + 1$, (3.18) is obtained if it is assumed that $(\tau_d - t[i+1]) = (\tau_d - t[i]) - 1$. ∎

In Figure 3.8 this model is applied taking $n = 2$ (globally quadratic) for three curves with different initial conditions a_0, a_1. Consequently, the resulting a_2 differs for each of the curves. Notice that all curves go to 0 at the same moment. The figure clearly shows the disadvantage of using motion profiles which are assumed to be globally quadratic. In order to bring the velocity "in step with" the position, the *first* deceleration that is applied can be very large. The deceleration is not constant along the trajectory, but due to a feedback control loop through which the trajectory is controlled, the first applied deceleration differs. This is mostly a disadvantage resulting in problems already mentioned above: Large acceleration differences are difficult to realize by mechanical structures (i.e., robots), and such large steps resulting from small changes in the input variables (the switching-on effect) are difficult to represent with continuous approximators such as feedforward neural networks. Clearly, an $n = 3$ system would be an improvement, but a more general approach is possible.

When choosing a value for n, this can be interpreted as the assumption that a_n will be constant along the trajectory; Figure 3.8 illustrates this for $n = 2$. The same figure also illustrates that $n = 2$ was not a good choice for those beginning conditions $\{a_0, a_1\}$: $a_{n=2}$ could not be kept constant along the whole trajectory. So, instead, we have to choose n based on the initial values of a_0 and a_1.

Varying-Order Polynomials

Given a moving system, the measured a_0/a_1 contains the information about the optimal trajectory that has to be followed until contact. With $a_0/a_1 =$

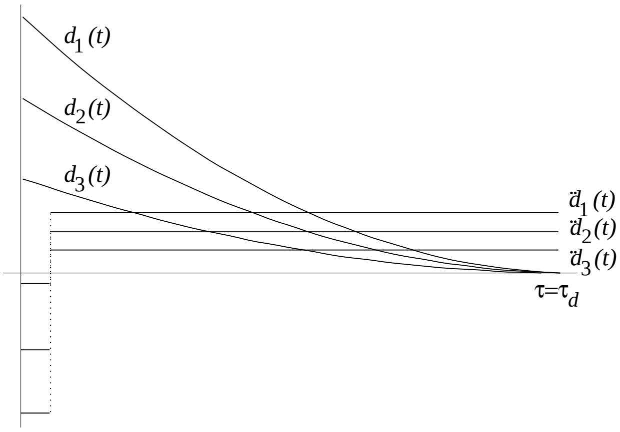

FIGURE 3.8. Time-dependent deceleration with an $n = 2$ system. Three globally quadratic trajectories are displayed. For each of the trajectories, the velocity $\dot{d}(t)$ at the beginning of deceleration is equal, but the distance from which is decelerated differs. The resulting decelerations (also shown, but not in scale) are different such that the trajectories all have the same τ_d. Note that after $i = 0$ (the first time interval), the decelerations are constant.

$-\tau_d/n$, $n > 0$, following a deceleration trajectory of order n renders the most smooth path to contact.

When we assume that the followed trajectory is *locally* quadratic, (3.18) can still be applied to calculate the desired deceleration. A deceleration algorithm, expressed in the desired a_2, could look like this:

1. Determine the desired order
$$n = \left\lceil -\frac{(\tau_d - t[i])a_1[i]}{a_0[i]} \right\rceil. \tag{3.21}$$
(We use the nearest integer although, strictly speaking, n need not be integer-valued.)

2. Determine the desired deceleration using (3.18).

3. Apply the $a_2[i]$ to the system.

4. Go to step 2.

Note that the algorithm can only be used when the calculated $n > 1$; for $n < 1$ the system must be first controlled to correct n.

Figure 3.9 shows the results of this algorithm applied to the same curves as in Figure 3.8. In this case, the deceleration does not make a single large step; instead, the position, velocity, and deceleration all smoothly decrease to 0.

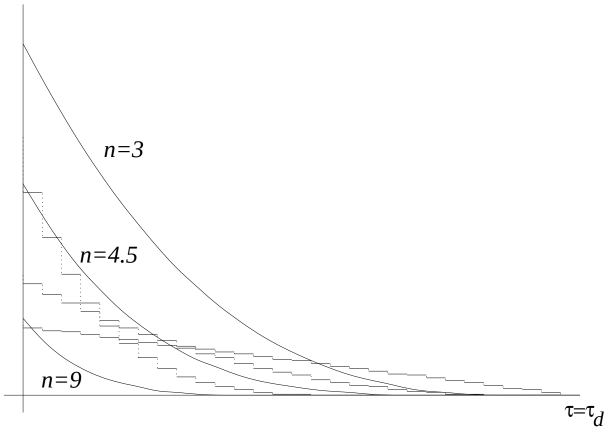

FIGURE 3.9. Time-dependent deceleration with a system where n is chosen automatically. For each of the trajectories, the velocity $\dot{d}(t)$ at the beginning of deceleration is equal, but the distance from which is decelerated differs. This time, the three trajectories have different $n = 3$, $n = 4.5$, and $n = 9$. The resulting decelerations (also shown) are different such that the trajectories all have the same τ_d ($n = 9$ and $n = 4.5$ appear to go to 0 earlier, but they remain positive until $t = \tau_d$).

3.5 Visual Measurement of the Stopping Criteria

The deceleration algorithms described above is expressed in parameters a_k. We know that the $d(t)$ and thus the a_k cannot be measured with one camera when there is no model at hand of the observed object(s).

Instead of looking at $d_x(t)$, $d_y(t)$, and $d_z(t)$, we have to consider the quantities that are actually measured: $\xi_x(t)$, $\xi_y(t)$, and $\xi_z(t)$. The first of these indicate the *observed* position of the object with respect of the center[3] of the camera image,[4] and the $\xi_z(t)$ is defined as

$$\xi_z(t) \equiv \frac{1}{\sqrt{\xi_A(t)}}. \tag{3.22}$$

Assuming a pinhole camera, the following model can be used for the observed area $\xi_A(t)$:

$$\xi_A(t) = \frac{f^2 A}{d_z(t)^2}, \tag{3.23}$$

such that

$$\xi_z(t) = \frac{d_z(t)}{f\sqrt{A}},$$

[3]Without loss of generality. The "center" can be defined anywhere in the image frame.

[4]This will generally be the number of pixels counted on the line between the center of the object and the center of the image frame.

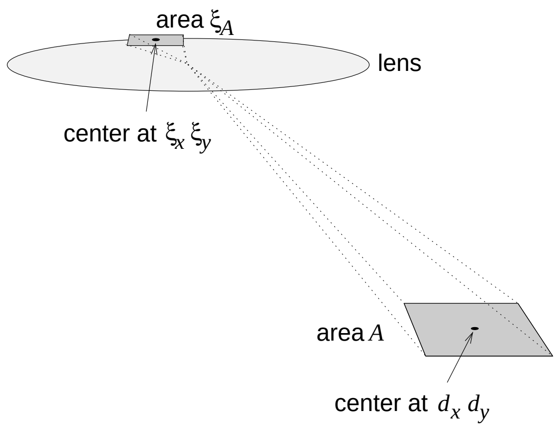

FIGURE 3.10. Exemplar mapping of an observed object in front of the lens on the CCD behind it. The object is placed at position $(120, 60, 100)$ "units" w.r.t. the lens, which has an $f = 10$. The area of the object is 50^2, such that the area of the observed object is 5^2.

where f is the focal length of the lens and A is the "real" area of the object. Since d_z and ξ_z are linearly related, the assumption that a second- or third-order polynomial fits $d_z(t)$ can also be used for $\xi_z(t)$.

Next, the models for $\xi_x(t)$ and $\xi_y(t)$ are investigated. If the camera-lens system is idealized by a pinhole camera, then the observed x and y positions $\xi_x(t)$ and $\xi_y(t)$ are (Figure 3.10)

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

Substituting (3.23),

$$\xi_x(t) = d_x(t)\sqrt{\frac{\xi_A(t)}{A}}, \tag{3.24}$$

and similarly for ξ_y. The observed x, y positions are linearly related to their Cartesian counterparts, but scaled with a factor $\sqrt{\xi_A(t)}$. Thus, if the Cartesian motion can be described by a second- (third)-order polynomial in the x, y, and z directions, this cannot be done in the observed visual domain ξ_x and ξ_y. This problem can be solved by scaling ξ_x and ξ_y by the same $\sqrt{\xi_A(t)}$, i.e.,

$$\xi'_x(t) = \frac{\xi_x(t)}{\sqrt{\xi_A(t)}}, \tag{3.25}$$

and similarly for ξ'_y. In conclusion, we find that

$$\xi'_x(t) = \frac{d_x(t)}{\sqrt{A}}, \qquad \xi'_y(t) = \frac{d_y(t)}{\sqrt{A}}, \qquad \xi_z(t) = \frac{d_z(t)}{f\sqrt{A}}. \tag{3.26}$$

The resulting $\xi'_x(t)$, $\xi'_y(t)$, and $\xi_z(t)$ are just as well fitted by the polynomial as x, y, and z, and from (3.5) follows

$$\xi(t) = \sum_{i=0}^{n} \upsilon_i t^i + o(t^n), \tag{3.27}$$

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

Once the parameters υ_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 (3.26), the υ_i's have a common constant

$$\upsilon_i = c a_i, \tag{3.28}$$

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

$$c_x = A^{-1/2}, \qquad c_y = A^{-1/2}, \qquad c_z = (f^2 A)^{-1/2}. \tag{3.29}$$

3.5.1 Measuring $\xi(t)$

Finally, we have to find the parameters υ_i. The method for finding these parameters is the same for ξ'_x, ξ'_y, and ξ_z.

In order to find the parameters of the polynomial expansions, least-mean-square optimization is used. Since the parameter dependency is linear, we can even revert to linear least squares and no iterative minimization method is required. Therefore we define the *chi-square merit function*,

$$\chi^2 = \sum_{i=1}^{M} \left(\frac{\xi(t_i) - \sum_{j=0}^{n} \upsilon_j t_i^j}{\sigma_i} \right)^2, \tag{3.30}$$

which is the quantity that must be minimized. M indicates the number of observations $\xi(t)$ that are available. As usual, σ_i is the standard deviation of $\xi(t_i)$. Also, the deviations of the υ_j are the diagonal elements C_{jj} of the covariance matrix C. Using the covariance matrix for the $n = 2$ case, we find

$$\sigma^2(\upsilon_0) = 3\frac{\sigma^2(3M^2 + 3M + 2)}{M(M-1)(M-2)}, \tag{3.31}$$

$$\sigma^2(\upsilon_1) = 12\frac{\sigma^2 M(8M + 11)(2M + 1)}{(M^2 - 1)(M^2 - 4)}. \tag{3.32}$$

Figure 3.11 shows the standard deviations $\sigma(\upsilon_0)$ and $\sigma(\upsilon_1)$ for several M when $\sigma^2 = 1$. From the figure it can be derived which M must be taken such as to obtain a certain accuracy in the parameters υ_0 and υ_1.

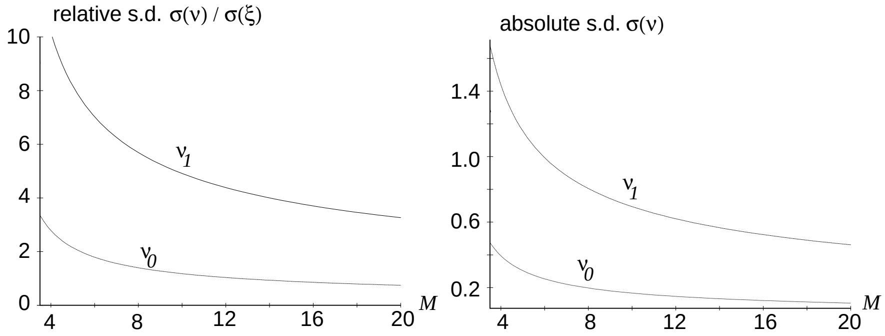

FIGURE 3.11. (Left) relative and (right) absolute standard deviation of v_j. The curves in the left figure express the standard deviations of v_0 and v_1 with respect to the standard deviation of the measurements. In the right figure are shown the calculated standard deviations when taking $\sigma^2(\xi_z) \approx 1/(0.05\sqrt{\xi_A})$ and $\xi_A = 20,000$ pixels^2.

3.6 Controlling the Manipulator

We have thus developed a method for determining constraints in the visual domain which lead to the desired goal state of rest in the visual domain. In this section we will show how a controller for a robot arm can be constructed which uses the above theory. The controller consists of two parts: (1) a system which generates trajectories in visual domain using the above theory; and (2) a neural network which translates the trajectory in visual domain to joint motor commands.

3.6.1 A Trajectory in the Visual Domain

How can the previously developed theory be used to construct a controller which decelerates a robot arm such that its hand-held camera "lands" on an object and comes to a standstill? In the deceleration method described above, the visual acceleration of the camera $v_2[i]$ is determined from its position $v_0[i]$ and velocity $v_1[i]$ relative to the object, and desired time to contact τ_d. So we can say that $v_2[i]$ is the *desired* visual acceleration necessary to satisfy the time-dependent deceleration criterion; hence we prefer to write $v_{2d}[i]$. During deceleration, this $v_{2d}[i]$ is applied to the system, which leads to a new $(v_0[i+1], v_1[i+1])$ in the next time step.

Why do we choose to control the acceleration $v_2[i]$? The construction of most industrial robots is such that the joints are controlled through constant torques, and therefore constant $\ddot{\theta}[i]$. Assuming small time intervals, a constant $\ddot{\theta}[i]$ approximately corresponds with a constant $v_2[i]$. We thus assume that the repeated approximations $d[i](t[i])$ by second-degree polynomials are precise enough when terms higher than the second order are ignored. The error introduced by this simplification is corrected for by refitting d at $i+1$, $i+2$, etc.

Instead of considering $v_{2d}[i]$ as the control variable for the controller, it is also possible to express a setpoint in a desired position and velocity at $i+1$. So we say that the next *setpoint* is a pair $(v_{0d}[i+1], v_{1d}[i+1])$, the elements of which are computed as

$$v_{0d}[i+1] = v_0[i] + v_1[i] + v_{2d}[i], \tag{3.33}$$

$$v_{1d}[i+1] = v_1[i] + 2v_{2d}[i], \tag{3.34}$$

where $v_{2d}[i]$ is computed with

$$v_{2d}[i] = -\frac{nv_0[i] + v_1[i](n + (\tau_d - t[i]) - 1)}{n + 2(\tau_d - t[i]) - 2}, \tag{3.35}$$

cf. (3.18), and n using

$$n = \left\lceil -\frac{(\tau_d - t[i])v_1[i]}{v_0[i]} \right\rceil, \tag{3.36}$$

cf. (3.21), assuming local quadratic polynomials for the trajectory $d(t[i])$ that is followed. Thus we end up with a system which has as inputs the measured $v_0[i]$, $v_1[i]$, and the remaining $(\tau_d - t[i])$, and as output the desired $v_{0d}[i+1]$ and $v_{1d}[i+1]$.

3.6.2 Translating the Trajectory to the Joint Domain

It has been shown that a trajectory can be determined in *visual* setpoints $(v_{0d}[i+1], v_{1d}[i+1])$ so as to satisfy the time-dependent constraint. The final question we have to ask is the following: How can we compute joint accelerations $\ddot{\theta}$ that make the robot-camera system follow those visual setpoints? The trajectory in the visual domain must be translated to a trajectory in the joint domain.

In the case that the area A of the object and focal length f of the camera are known constants, then the relationship between v_i and a_i is known. In that case the trajectory $(v_{0d}[i+1], v_{1d}[i+1])$ in the visual domain can be calculated from the measured trajectory in the visual domain, and with known kinematics of the robot this can be used to compute a trajectory in the joint domain. We know that the mapping

$$a_0[i], a_1[i], a_{0d}[i+1], a_{1d}[i+1], \theta[i], \dot{\theta}[i], \ddot{\theta}[i] \rightarrow \ddot{\theta}[i+1] \tag{3.37}$$

is a function (i.e., can be computed) for the given robot-camera system.

Of course, in our case the relationship between v_i and a_i is *not* known, and it even varies with the size of the object. The question is whether the mapping

$$v_0[i], v_1[i], v_{0d}[i+1], v_{1d}[i+1], \theta[i], \dot{\theta}[i], \ddot{\theta}[i] \rightarrow \ddot{\theta}[i+1] \tag{3.38}$$

is a function *also*, i.e., does not have multiple solutions.

The reason that this is so is that the end effector velocity can be measured (albeit in joint space), such that the visual measurements are disambiguated. A mathematical argument proceeds as follows. From the inverse kinematics we know that a constant function $\dot{\mathbf{d}}(\theta, \dot{\theta}) = (\dot{d}_x(\theta, \dot{\theta}), \dot{d}_y(\theta, \dot{\theta}), \dot{d}_z(\theta, \dot{\theta}))$, changing only with the kinematics of the robot, exists with which the velocity of the end effector in world space can be computed from the θ and $\dot{\theta}$. Recalling the relationship between v_i and a_i in (3.29), with the (unknown yet constant) inverse kinematics of the robot we have an expression for a_1 in terms of $\dot{\mathbf{d}}(\theta, \dot{\theta})$ in the joint domain, while we can measure v_{1xyz} with the camera. Therefore we have an expression for A and f:

$$A = \frac{{a_{1x}}^2}{{v_{1x}}^2} = \frac{\dot{d}_x(\theta, \dot{\theta})^2}{{v_{1x}}^2}, \qquad f = \frac{a_{1z}}{\sqrt{A} v_{1z}} = \frac{\dot{d}_z(\theta, \dot{\theta})}{v_{1z} \dot{d}_x(\theta, \dot{\theta}) / v_{1x}}. \tag{3.39}$$

Therefore we conclude that the left-hand sides of (3.37) and (3.38) carry the same information. The mapping (3.38), even though it cannot be computed without knowledge of the robot's inverse kinematics, can be learned using a neural network. We will show in the next section how this is done.

3.6.3 Controller Structure

The resulting controller consists of three parts:

1. The *interpolator* uses the measured $\xi(t)$ and $\theta(t)$ to determine the joint positions $\theta[i]$ and their derivatives $\dot{\theta}[i]$ and $\ddot{\theta}[i]$, as well as the visual trajectory parameters $v_j[i]$.
2. The visual trajectory parameters are input to the *criterionator*, which applies equations (3.35), (3.36), (3.33), and (3.34) in that order to calculate the next visual setpoint $(v_{0d}[i+1], v_{1d}[i+1])$.
3. The current visual measurement $(v_0[i], v_1[i])$, the visual setpoint, and $\theta[i]$, $\dot{\theta}[i]$, and $\ddot{\theta}[i]$ are input to the neural network.
4. The neural network generates a joint acceleration $\ddot{\theta}[i+1]$ which is sent to the robot.
5. The robot-camera system closes the control loop.

Figure 3.12 shows the components of the time-to-contact controller in the control loop. Note that the criterionator, in order to determine $(\tau_d - t[i])$, needs a clock.

Implementational Details

The neural network consists of a separate control process called "connel" and a learning process called "lummel." These two processes cooperate as follows. Each newly available learning sample is sent from connel to lummel, which adds it to its bin of recent learning samples. Each time before a forward propagation through the feedforward neural network is done in order to compute the next joint acceleration, a weight matrix is sent from lummel to connel.

3.6.4 Generating Learning Samples

It is imperative that the learning samples be generated online. We investigate which data are necessary to create learning samples which describe the desired behavior of the neural network $\mathcal{N}$, and first investigate the functions that are performed by the neural network and the robot-camera system.

The Neural Network

The input to the neural network consists of the desired trajectory in visual setpoints (measured visual state at i and desired visual state at $i+1$), in conjunction with the state of the robot. The output of the controller consists of joint accelerations which must be applied to the robot to instantiate the supplied trajectory:

$$\mathcal{N}\left(v_0[i], v_1[i], v_{0d}[i+1], v_{1d}[i+1], \theta[i], \dot{\theta}[i], \ddot{\theta}[i]\right) = \ddot{\theta}[i+1], \qquad (3.40)$$

where $v_k[j]$ are separately input for the x, y, and z components. The total number of inputs to $\mathcal{N}(\,)$ equals 20.

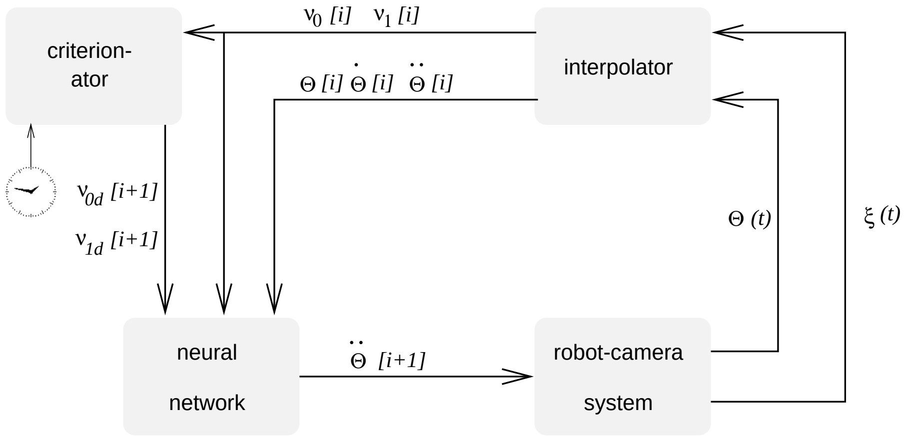

FIGURE 3.12. The structure of the time-to-contact control loop.

The Robot-Camera System

The robot, observing a stationary object, performs the following transition:

$$\mathcal{R}\left(v_0[i], v_1[i], \theta[i], \dot{\theta}[i], \ddot{\theta}[i+1]\right) = \left(v_0[i+1], v_1[i+1], \theta[i+1], \dot{\theta}[i+1]\right). \tag{3.41}$$

In this case we take $v_0[i]$ and $v_1[i]$ as inputs of $\mathcal{R}$ since they describe the position of the object w.r.t. the moving hand-held camera.

A Learning Sample

From the two above transitions it is concluded that a valid learning sample is constructed as follows:

$$\mathcal{C}^0\left(v_0[i], v_1[i], v_0[i+1], v_1[i+1], \theta[i], \dot{\theta}[i], \ddot{\theta}[i]\right) = \ddot{\theta}[i+1]$$

where $\mathcal{C}^0$ indicates the ideal controller. The neural network thus approximates a function of the inverse kinematics of the robot-camera system, where the visual observations are scaled by a "varying constant" c. In effect, learning by interpolation is used.

3.6.5 Experimental Procedure

The exact experimental procedure is as follows.

1. The target object, which has randomly chosen dimensions, is placed at a random position in the reach space of the robot (see Figure 3.13). Starting from a random initial joint position, the robot is given random joint accelerations for joints 1, 2, and 3.

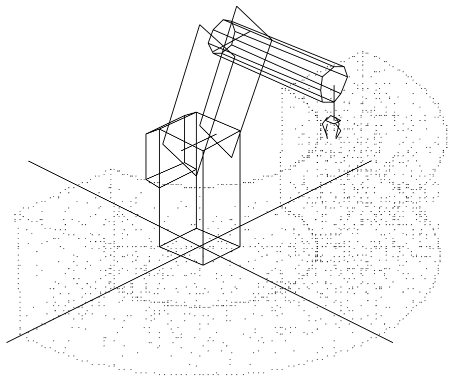

FIGURE 3.13. Reach space of the robot used in the simulator experiments. The cloud of dots marks the positions where target objects can be placed.

2. Choose a desired time to contact $(\tau_d - t[0])$ (in the simulator experiments it was randomly chosen between 10 and 40 simulator time steps) and set $i = 0$.

3. The learning algorithm is started:

   (a) Measure $(\upsilon_0[i], \upsilon_1[i])$ for the x, y, and z directions, and measure $\theta[i]$, $\dot{\theta}[i]$, and $\ddot{\theta}[i]$ for joints 1, 2, and 3.

   (b) If $i > 0$ a learning sample,

   $$\left(\upsilon_0[i], \upsilon_1[i], \theta[i-1], \dot{\theta}[i-1], \ddot{\theta}[i-1]\right) \Rightarrow \left(\ddot{\theta}[i]\right),$$

   is created and added to the set of learning samples.

   (c) If $(\tau_d - t[i]) \leq 0$, go to step 4;

   (d) $\upsilon_{0d}[i+1]$ and $\upsilon_{1d}[i+1]$ are determined using the criterionator.

   (e) The network requests a new weight matrix from lummel, and generates a $\ddot{\theta}[i+1]$ for joints 1, 2, and 3, which is sent to the robot.

   (f) $(\tau_d - t[i+1]) = (\tau_d - t[i]) - 1$. $i \leftarrow i + 1$; go to step 3a.

4. Go back to step 1.

3.6.6 Exploration

Naturally, the self-learning controller can never learn what it has not seen. Some method of exploration of the input space and output space is required. This is done as follows. The neural controller is used to approach the target. When the final distance between the target and the end effector exceeds some threshold (10 cm in the simulator experiments), the test is performed again with the same initial robot position and object position, but the output of the neural network is augmented with additive white noise. This is repeated up to three times with increasing noise levels.

3.6.7 Finding the Optimal Network

We use the theory presented by Vyšniauskas *et al.* [10] to determine the optimal network size and number of learning samples for this problem. Using this theory, we are interested in the asymptotic behavior of the error in approximation ϵ_a as a function of the number of learning samples N and number of hidden units κ for $1 \ll \kappa \ll \kappa_*$ and $1 \ll N \ll N_*$. According to Vyšniauskas *et al.* [10], ϵ_a can be approximated by

$$\epsilon_a^2(N, \kappa) = \sum_p \sum_{i+j=p} \gamma_{ij} (N^{-1} - N_*^{-1})^i (\kappa^{-1} - \kappa_*^{-1})^j \approx \sum_p \sum_{i+j=p} \frac{\gamma_{ij}}{N^i \kappa^j}, \tag{3.42}$$

where $i, j, p \in \aleph$ and γ_{ij} are the parameters we are looking for. These parameters can be found by minimizing the difference between the error in approximation given by the feedforward neural network as a function of κ and N, and (3.42). We call the approximation the AMEF($p_{\min}, p_{\max}$) function.

The procedure for this is as follows.

1. A total of 5,500 samples are generated.
2. For different numbers of hidden units κ and learning samples N, perform the minimization procedure and note the resultant summed squared error. In this case we take $5 \leq \kappa \leq 40$ and $100 \leq N \leq 1500$. A total of 4,000 separate samples are used for cross-validation.
3. Compute the AMEF($p_{\min}$, $p_{\max}$) (as described in [10]) function to fit the data obtained in the previous step, and determine the optimal parameters $p_{\min}$, $p_{\max}$, and γ. In this case, $p_{\min} = 1$ and $p_{\max} = 4$ was taken.
4. The resulting model can be used to find the minimal resources line (i.e., where computational resources are minimal to obtain a certain error).

However, it appears that the AMEF( ) function does not fit the data very well. We follow a suggestion by Vyšniauskas ([9], see also [11]) and change the expansion of the AMEF( ) function of equation (3.42) to

$$\epsilon_a(N, \kappa) \approx \sum_p \sum_{i+j=p} \frac{\gamma_{ij}}{N^{i/2}\kappa^j}. \tag{3.43}$$

The dependency of ϵ_a on N is thus also in accordance with Barron [1]. The above steps are repeated for this adapted AMEF( ) function, leading to $p_{\min} = 1$ and $p_{\max} = 4$. The results of these experiments are shown in Figure 3.14. The figures show the approximation by the adapted AMEF(1, 4) function for the estimated γ_{ij} of the robot-camera mapping function as defined by the learning samples for a network with 2, 4, 6, and 8 hidden units, for increasing numbers of learning samples. Apparently, for a large number of hidden units the neural network *over*fits the data when the number of hidden units increases; although the error for the learning samples decreases, the error for the test samples does not always decrease when more hidden units are taken.

As before, to find the network with minimal resources, i.e., the network with a certain desired accuracy ϵ^0 for which the computational complexity

$$r \sim N\kappa$$

is minimal, the adapted AMEF(1, 4) model must be evaluated such that $\epsilon_a(N, \kappa) = \epsilon^0$. Figure 3.15 shows the minimal resources line for the time-

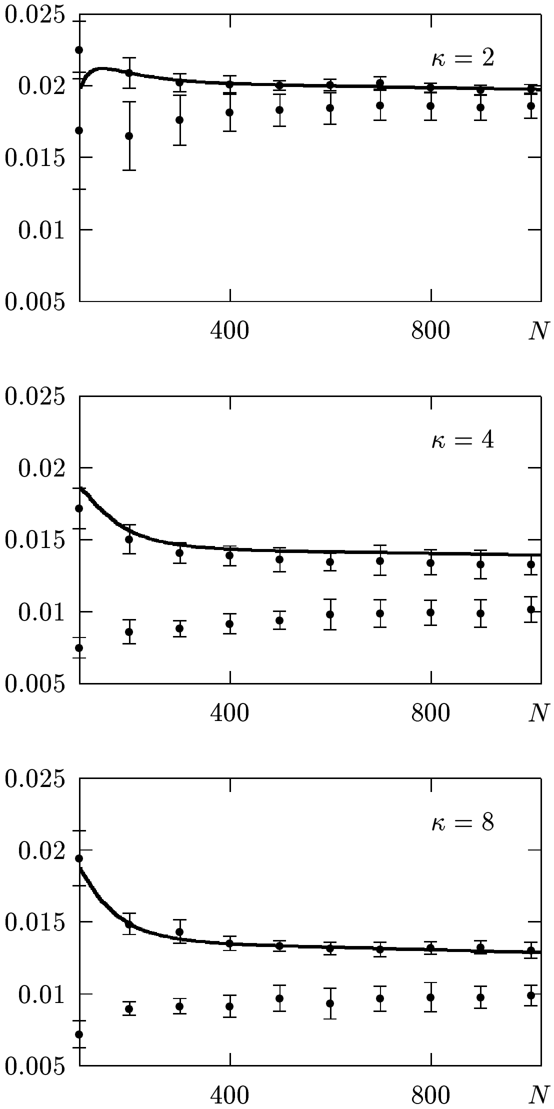

FIGURE 3.14. Error graphs with model: Accuracy in approximation of the camera-robot mapping by the feedforward network with 2, 4, 6, and 8 hidden units. The learning error is represented by error bars at the bottom, the generalization error by error bars at the top, the adapted AMEF(1, 4) is plotted with the thick solid line. On the vertical axes is printed the error in the network output (joint acceleration) in $^\circ/\mathrm{tick}^2$, where a tick indicates a simulator time unit. Note that the AMEF(1, 4) model is not very successful for small N and κ.

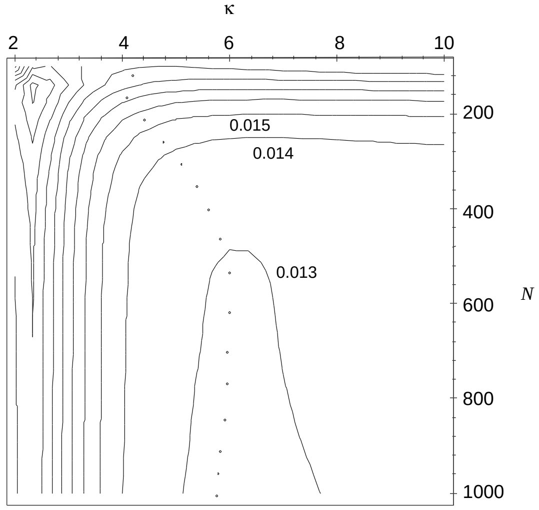

FIGURE 3.15. Minimal resources for the time-to-contact neural network. The figures show the contour lines of the adapted AMEF(1, 4) model, as well as the minimal resources (dots) for this problem. Each contour line indicates an increase in the error of 0.001. As the figure shows, it is not useful to take more than 6 hidden units in the neural network; this will lead to an expected average error in the network output of just below $0.013°/\text{tick}^2$, where a tick indicates a simulator time unit. Note that the minimal resources line makes a wrong prediction for large N.

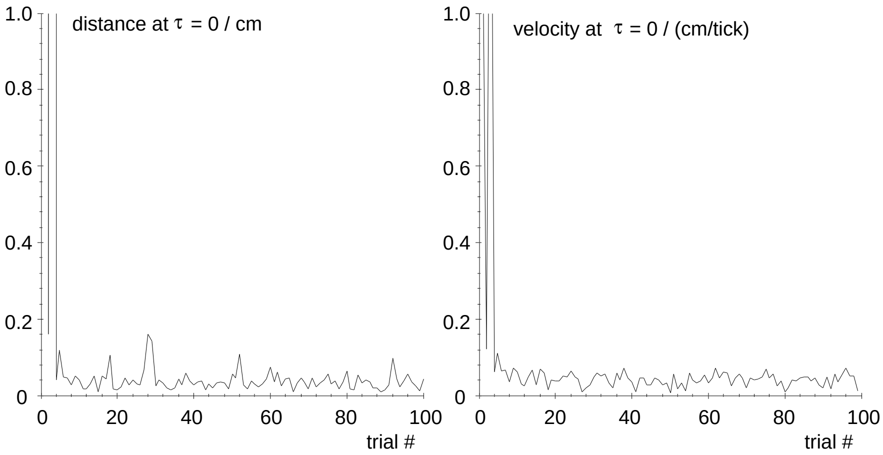

FIGURE 3.16. Distance and velocity at $\tau = 0$. The left figure shows the distance $\sqrt{d(\tau_d = 0)_x^2 + d(\tau_d = 0)_y^2 + d(\tau_d = 0)_z^2}$ between the end effector and the approached object. The right figure shows the velocity $\sqrt{\dot{d}(\tau_d = 0)_x^2 + \dot{d}(\tau_d = 0)_y^2 + \dot{d}(\tau_d = 0)_z^2}$ 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.

to-contact robot control neural network. The figure clearly shows that a network within the order of 5–6 hidden units is optimal, and that the number of learning samples is not very relevant when chosen larger than approximately 400. This graph was used to choose the optimal network and learning set size as follows. If it is assumed that the relationship between the visual acceleration υ_2 and joint acceleration $\ddot{\theta}$ is locally linear, then the error of the network output in Figure 3.14 can be coarsely interpreted as the error in visual acceleration. Thus it was determined that in this case, a network with 6 hidden units and at least 400 learning samples is optimal. This network size was used in the experiments.

3.7 Results

In order to measure the success of the method when 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 $\mathbf{d}(t) = \dot{\mathbf{d}}(t) = 0$ 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 3.16. 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, four), the positional error at $\tau_d = 0$ is below 1 mm, while the velocity is very low.

To illustrate the followed trajectories, the velocity in the x, y, and z

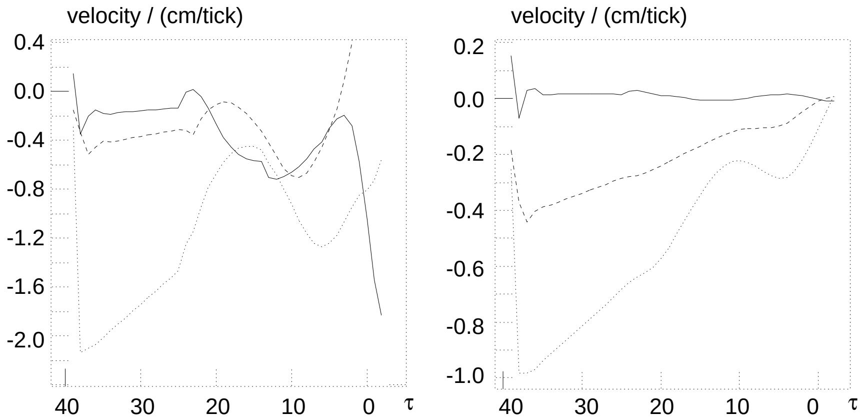

FIGURE 3.17. Two exemplar trajectories followed with time-to-contact control. Both figures show the velocity $\dot{d}(\tau_d)_x$ (solid), $\dot{d}(\tau_d)_y$ (dashed), and $\dot{d}(\tau_d)_z$ (dotted). In both figures, the order n of the deceleration was chosen to be 2. (Left) Trial 2 is not successful. The velocities vary greatly during the trajectory, and do not go to 0. (Right) Trial 40 is successful. The velocities in the x, y, and z directions decay smoothly to 0.

directions for trial 2 (not successful) and trial 40 (successful) are plotted against τ in Figure 3.17.

3.7.1 Influence of Input Noise

The influence of noise on the input to the network on the results is experimentally determined. Thereto we use the noise estimates for typical cases, and see how the addition of such noise to the network input influences the results of the deceleration.

To be exact, the following noise levels are added. We take as standard deviations (which were measured from the real robot)

$$\sigma_x = 0.010\sqrt{\xi_A}, \quad \sigma_y = 0.016\sqrt{\xi_A}, \quad \sigma_A = 0.0036\sqrt{\xi_A}.$$

Furthermore, the x and y positions are computed by taking the average position of the number of pixels of the measured area, such that the σ_x and σ_y both must be divided by ξ_A (which is measured in pixels).

Next we have to find the standard deviation of the z component of motion. It is shown in Appendix 3.C that this z component also has a normal distribution when A is a normally distributed random variable. In fact, ξ_z is normal distributed with mean $1/\sqrt{\overline{\xi_A}}$ and variance $\sigma_A^2/(2\overline{\xi_A}\sqrt{\overline{\xi_A}})$.

Finally, we have to take into account the relationships (3.31) and (3.32). The resulting standard deviations are

$$\sigma(v_{0x}) = 2 \cdot 0.010/\sqrt{\xi_A}, \quad \sigma(v_{1x}) = 7 \cdot 0.010/\sqrt{\xi_A},$$

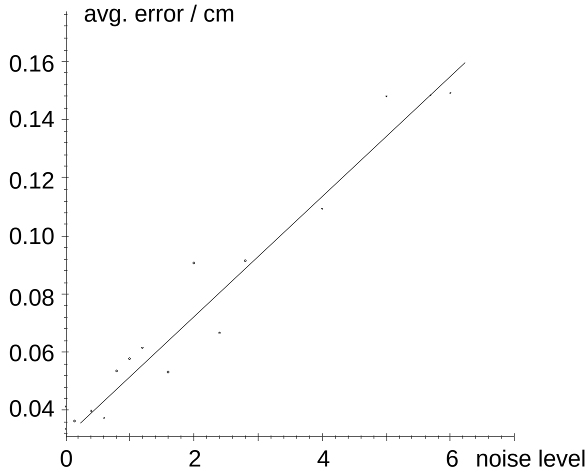

FIGURE 3.18. Results of the simulation when noise in the visual input is applied. The figure shows the distance $\sqrt{d(\tau=0)_x^2 + d(\tau=0)_y^2 + d(\tau=0)_z^2}$ between the end effector and the approached object averaged over 200 goals (marked by dots on the graph). The vertical axis shows the 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 so 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 2.0 cm; at $l = 8$ it is as high as 4.0 due to overshoots.

$$\sigma(\upsilon_{0y}) = 2 \cdot 0.016/\sqrt{\xi_A}, \quad \sigma(\upsilon_{1y}) = 7 \cdot 0.016/\sqrt{\xi_A},$$
$$\sigma(\upsilon_{0z}) = 2\xi_A/(2\overline{\xi}_A\sqrt{\overline{\xi}_A}), \quad \sigma(\upsilon_{0z}) = 7\xi_A/(2\overline{\xi}_A\sqrt{\overline{\xi}_A}).$$

where l is a noise-level factor. These standard deviations are used in the computations.

Note that the standard deviation of the joint measurements is taken to be 0. More important, the positioning inaccuracy of the robot is not taken into account. This positioning inaccuracy would not, however, influence the behavior of the neural network.

Figure 3.18 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 seven times the expected amount of noise.

3.8 Discussion

A method has been proposed which can control the end effector of a monocular robot in three-dimensional space by using optical flow feedback to obtain depth information. The system does not use knowledge of the robot arm or camera, and does not require any calibration at all. Instead, from the visual feedback, setpoints in the visual domain are calculated, and these setpoints are translated into the joint domain using an adaptive neural controller.

Results show that this system can be used to create a robust feedback robot arm control system. The error in the simulator experiments is very low: After only a few trials, the grasping distance at the desired time to contact is around 0.05 cm, while the gripper velocity is nearly 0. The system has also been shown to remain stable under the influence of noise in the input parameters and learning samples.

The choice of the controller input signals, by requiring only first-derivative visual information, makes the system very viable for implementation on a real robot arm. However, a few questions remain to be asked.

The Moment of "Contact"

The moment of "contact" where $d_z = 0$, is that moment where the object is placed at distance f from the (pinhole) lens. But what happens when we want to place this final point farther away from or closer to the lens? Changing the position where $d_z = 0$ can best be done by changing the optical system to get a different focal distance; after all, by decoupling the x, y, and z components of the visual observation, only the latter depends on the focal distance. Alternatively, the time-dependent deceleration constraint (3.17) can be augmented with an offset constant, depending on the actual size A of the object. On the downside, this would mean that the resulting distance would be different for different size objects.

Acceleration Toward the Object

In this chapter we have concentrated on the deceleration of the manipulator. Naturally, the acceleration is also of importance, yet no absolute positional information of the object is available when the camera is not moving. Therefore, the only way to solve the problem is to assume a quasi-random motion, and correct from that moment as soon as the first visual measurements are available. This drawback is, in fact, not untypical for various biological systems.

Extension to the Controller

In the controller described above, the values of $v_{0d}[i+1]$ and $v_{1d}[i+1]$ are calculated using the criterionator. However, this can be replaced by a neural network. This neural network $\mathcal{N}'$ has to generate the following transition:

$$\mathcal{N}'\left(v_0[i], v_1[i], (\tau - t[i+1])\right) = \left(v_{0d}[i+1], v_{1d}[i+1]\right).$$

After the transitions (3.40) and (3.41), the actually reached $v_0[i+1]$ and $v_1[i+1]$ are available. The transition from $v_0[i]$ and $v_1[i]$ to these signals at $i+1$ contains the necessary information on which transitions are reasonably performed by the robot; these samples thus form the learning samples of the robot. Second, $\mathcal{N}'$ should always map those inputs where $(\tau - t[i+1]) = 0$ on $(0,0)$. Finally, the resultant "grasping error," i.e., the remaining v_0 and v_1 when $\tau = 0$, can be used as the variance of the learning samples generated for $\mathcal{N}'$ along the trajectory leading to that grasping error.

3.A Derivation of the Stopping Criteria

Theorem

A signal $d(t)$ is approximated by a finite Taylor expansion,

$$d(t) = \sum_{i=0}^{n} a_i t^i, \tag{3.44}$$

where $a_n \neq 0$. Then the boundary conditions

$$\forall 0 \leq k < n : \quad d(\tau)^{(k)} = 0 \tag{3.45}$$

imply

$$\frac{a_n^{n-k-1} a_k}{a_{n-1}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}. \tag{3.46}$$

Proof

Using the Taylor expansion for $d(t)$, the kth derivative of $d(t)$ equals

$$d(t)^{(k)} = \sum_{i=k}^{n} \frac{i!}{(i-k)!} a_i t^{i-k}. \tag{3.47}$$

Therefore, the constraints (3.45) can be rewritten as

$$\forall 0 \leq k < n : \quad \sum_{i=k}^{n} \frac{i!}{(i-k)!} a_i \tau^{i-k} = 0. \tag{3.48}$$

If we first look at the constraint for $k = n-1$, then

$$(n-1)!\, a_{n-1} + n!\, a_n \tau = 0 \tag{3.49}$$

or, since $a_n \neq 0$,

$$\tau = -\frac{a_{n-1}}{n a_n}. \tag{3.50}$$

Second, if it can be shown that for all $0 \le h < n$,

$$a_h = \binom{n}{h}(-\tau)^{n-h} a_n, \tag{3.51}$$

then, by combining (3.50) and (3.51), we find that

$$\frac{a_n^{n-k-1} a_k}{a_{n-1}^{n-k}} = \frac{1}{n^{n-k}} \binom{n}{k}. \tag{3.52}$$

To prove assumption (3.51), first observe that, with the equation for τ in (3.50), it is clearly true for $h = n-1$. Now let us assume that equation (3.51) is true for all a_i with $n > i \ge h$. By multiplying both sides of equation (3.48) by $1/h!$ we find that

$$0 = \sum_{i=h}^{n} \binom{i}{h} a_i \tau^{i-h}. \tag{3.53}$$

Similarly for $k = h - 1$,

$$\begin{aligned} 0 &= \sum_{i=h-1}^{n} \binom{i}{h-1} a_i \tau^{i-(h-1)} \\ &= a_{h-1} + \sum_{i=h}^{n} \binom{i}{h-1} a_i \tau^{i-(h-1)}. \end{aligned} \tag{3.54}$$

Using the assumption (3.51) for a_i with $i \ge h$,

$$\begin{aligned} 0 &= a_{h-1} + \sum_{i=h}^{n} \binom{i}{h-1} \binom{n}{i} \tau^{n-i} (-1)^{n-i} a_n \tau^{i-(h-1)} \\ &= a_{h-1} + a_n \tau^{n-(h-1)} \left[\sum_{i=h}^{n} (-1)^{n-i} \binom{i}{h-1} \binom{n}{i} \right]. \end{aligned} \tag{3.55}$$

The factor between brackets is equal to $\binom{n}{h-1}(-1)^{n-h}$, which we will show below, such that

$$a_{h-1} = \binom{n}{h-1}(-\tau)^{n-(h-1)} a_n. \tag{3.56}$$

By induction we prove equation (3.51) for all $0 \le h < n$.

Now we only need to prove the following.

Lemma

For positive integer n and positive integer $k \le n$,

$$\sum_{i=k}^{n} (-1)^{n-i} \binom{i}{k-1} \binom{n}{i} = \binom{n}{k-1} (-1)^{n-k}. \tag{3.57}$$

Proof

If we introduce new variables $j = i - k$ and $m = n - k$, then

$$\begin{aligned}
&\sum_{i=k}^{n}(-1)^{n-i}\binom{i}{k-1}\binom{n}{i} \\
&= \sum_{j=0}^{m}(-1)^{m-j}\binom{j+k}{k-1}\binom{m+k}{j+k} \\
&\text{(since } \tbinom{a}{b}\tbinom{b}{c} = \tbinom{a}{c}\tbinom{a-c}{b-c}\text{)} \\
&= (-1)^{m}\binom{m+k}{k-1}\sum_{j=0}^{m}(-1)^{j}\binom{m+1}{j+1} \\
&= -(-1)^{n-k}\binom{n}{k-1}\left[\sum_{j=0}^{m+1}(-1)^{j}\binom{m+1}{j} - (-1)^{0}\binom{m}{0}\right] \\
&\text{(using the binomial theorem for } (a+b)^{m+1} \\
&\text{with } a = -1 \text{ and } b = 1\text{)} \\
&= -(-1)^{n-k}\binom{n}{k-1}(0-1) \\
&= \binom{n}{k-1}(-1)^{n-k},
\end{aligned}$$

which completes the proof. ■

3.B Proof of Theorem 3.2

In this section, we will write a_j for $a_j[0]$.

Theorem

Using the equivalence of $d[i]^{(k)}$ and $a_k[i]$, we have to show that, if

$$\forall i: \quad \frac{d[i](t[i])}{d'[i](t[i])} = \frac{-(\tau_d - t[i])}{n}, \tag{3.58}$$

where

$$d[i](t[i]) = \sum_{j=0}^{n} a_j t[i]^j \quad \text{and} \quad d'[i](t[i]) = \sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^j, \tag{3.59}$$

then

$$\frac{a_k}{a_{n-1}} = (-\tau_d)^{n-k-1}\frac{1}{n}\binom{n}{k}, \qquad \text{for } 0 \le k < n. \tag{3.60}$$

Proof

Substituting (3.59) in (3.58), we get

$$\begin{aligned}\forall i: \quad \sum_{j=0}^{n} n a_j t[i] &= (t[i]-\tau_d)\sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^j \\ &= \sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^{j+1} - \sum_{j=0}^{n-1}\tau_d(j+1)a_{j+1}t[i]^j \\ &= t[i]\sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^j - \tau_d\sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^j,\end{aligned}$$

such that

$$\forall i: \quad n a_0 + \sum_{j=1}^{n} n a_j t[i]^j - \sum_{j=1}^{n} j a_j t[i]^j + \tau_d \sum_{j=0}^{n-1}(j+1)a_{j+1}t[i]^j = 0,$$

or

$$\begin{aligned}\forall i: \quad & n a_0 + (n a_n - n a_n)t[i]^n \\ & + \sum_{j=1}^{n-1}\left[(n-j)a_j + (j+1)\tau_d a_{j+1}\right]t[i]^j + \tau_d a_1 t[i]^0 = 0.\end{aligned}$$

Therefore, it follows that

$$\forall j, 0 \le j < n: \quad (n-j)a_j = -\tau_d(j+1)a_{j+1},$$

which can be written as

$$\forall j, 0 \le j < n: \quad \frac{a_{j+1}}{a_j} = -\frac{n-j}{\tau_d(j+1)}.$$

This solution is equivalent to (3.60), which proves the theorem. ■

3.C Nonlinear Transform of a Noisy Signal

In the previous section, it was assumed that the measurements $\xi(t)$ are normally distributed. How reasonable is this assumption? After all, by fitting measurements $\xi_z(t)$ instead of the measured area $\boldsymbol{\xi}_A(t)$, the noise in the original measurement is also transformed.

Recall that $\xi_z(t)$ is defined as

$$\xi_z(t) \equiv \frac{1}{\sqrt{\boldsymbol{\xi}_A(t)}}. \tag{3.61}$$

In general, the probability distribution $P(\xi_z)$ of $\xi_z = f(\xi_A)$ can be derived from the probability distribution $P(\xi_A)$ of ξ_A as

$$\begin{aligned} P(\xi_z) &= \int_{\xi_A} \delta(\xi_z - f(\xi_A))P(\xi_A)d\xi_A \\ &= P\left(f^{-1}(\xi_z)\right)\left|\frac{df}{d\xi_z}\left(f^{-1}(\xi_z)\right)\right|. \end{aligned}$$

It is reasonable to assume a normal distribution for ξ_A, such that the probability distribution of ξ_A is

$$P(\xi_A) = \frac{1}{\sqrt{2\pi\sigma_A^2}} \exp\left[-\frac{(\xi_A - \overline{\xi_A})^2}{2\sigma_A^2}\right]. \quad (3.62)$$

The resulting distribution for ξ_z is then

$$\begin{aligned} P(\xi_z) &= \frac{2}{\xi_z^3\sqrt{2\pi\sigma_A^2}} \exp\left[-\frac{((1/\xi_z^2) - \overline{\xi_A})^2}{2\sigma_A^2}\right] \\ &= 2\xi_A\sqrt{\xi_A}P(\xi_A). \end{aligned}$$

This is not a normal distribution anymore. With knowledge of the standard deviation of ξ_A, however, it can be shown to be approximately normal. If the first three terms of the Taylor series for $\xi_z(t)$ are considered,

$$\xi_z(t) \approx \frac{1}{\sqrt{\overline{\xi_A}}} + \frac{1}{2\overline{\xi_A}\sqrt{\overline{\xi_A}}}|\overline{\xi_A} - \xi_A| + \frac{3}{4\overline{\xi_A}^2\sqrt{\overline{\xi_A}}}(\overline{\xi_A} - \xi_A)^2. \quad (3.63)$$

If the second term of this Taylor series is "sufficiently small" in comparison with the first, then ξ_z can be approximated by a linear function of ξ_A, such that ξ_z is indeed approximately normal distributed with mean $1/\sqrt{\overline{\xi_A}}$ and variance $\sigma_A^2/2(\overline{\xi_A}\sqrt{\overline{\xi_A}})$. It has been measured that $|\overline{\xi_A} - \xi_A| \approx 0.0036\sqrt{\overline{\xi_A}}$, such that the quadratic term in (3.63) is approximately $150\sqrt{e}$ times smaller than the linear term. With a typical circumference of 200 pixels, the quadratic term will be over a factor 25,000 smaller, and can be safely ignored.

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

References

[1] A. R. Barron. Approximation and estimation bounds for artificial neural networks. In *Proceedings of the Fourth Annual Workshop on Computational Learning Theory*, pages 243–249, 1991.

[2] R. Cipolla and A. Blake. Surface orientation and time to contact from image divergence and deformation. In G. Sandini, editor, *Computer Vision—ECCV '92*, pages 187–202. Springer-Verlag, Berlin, 1992.

[3] J. J. Gibson. *The Perception of the Visual World.* Houghton Mifflin, Boston, 1950.

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

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

[6] G. R. Martin. Form and function in the optical structure of bird eyes. In P. Green and M. Davies, editors, *Perception and Motor Control in Birds.* Springer-Verlag, Berlin, 1993.

[7] R. Sharma. Active vision in robot navigation: Monitoring time-to-collision while tracking. In *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2203–2208. IEEE, June 1992.

[8] D. Vernon and M. Tistarelli. Using camera motion to estimate range for robotic parts manipulation. *IEEE Transactions on Robotics and Automation*, 7(5):509–521, October 1990.

[9] V. Vyšniauskas. Personal communication, 1994.

[10] V. Vyšniauskas, F. C. A. Groen, and B. J. A. Kröse. The optimal number of learning samples and hidden units in function approximation with a feedforward network. Technical Report CS–93–15, Department of Computer Systems, University of Amsterdam, Amsterdam, The Netherlands, 1993.

[11] S. S. Wilks. *Mathematical Statistics.* Wiley, New York, 1962.