ELSEVIER

Neurocomputing 37 (2001) 51–70

NEUROCOMPUTING

www.elsevier.com/locate/neucom

Reinforcement learning in a rule-based navigator for robotic manipulators

Kaspar Althoefer[a,*], Bart Krekelberg[b], Dirk Husmeier[c], Lakmal Seneviratne[a]

[a]*Department of Mechanical Engineering, King's College London, Strand, London WC2R 2LS, UK*
[b]*Department of Neurobiology ND7/30, Ruhr University Bochum, 44780 Bochum, Germany*
[c]*Biomathematics and Statistics Scotland (BioSS), SCRI, Invergowrie, Dundee DD2 5DA, UK*

Received 9 August 1999; accepted 13 April 2000

Abstract

This paper reports on a navigation system for robotic manipulators. The control system combines a repelling influence related to the distance between manipulator and nearby obstacles with the attracting influence produced by the angular difference between actual and final manipulator configuration to generate actuating motor commands. The use of fuzzy logic for the implementation of these behaviors leads to a transparent system that can be tuned by hand or by a learning algorithm. The proposed learning algorithm, based on reinforcement-learning neural network techniques, can adapt the navigator to the idiosyncratic requirements of particular manipulators, as well as the environments they operate in. The navigation method, combining the transparency of fuzzy logic with the adaptability of neural networks, has successfully been applied to robot arms in different environments.

Keywords: Reinforcement learning; Behavior; Fuzzy logic; Navigation; Robotic manipulator

1. Introduction

In everyday life, we move effortlessly through rooms filled with large numbers of odd-shaped obstacles. We find a chair to sit on, a cup to drink from — all without

* Corresponding author. Tel.: + 44-20-7848-2431; fax: + 44-20-7848-2932.
E-mail addresses: k.althoefer@kcl.ac.uk (K. Althoefer), bart@neurobiologie.ruhr-uni-bochum.de (B. Krekelberg), dirk@bioss.sari.ac.uk (D. Husmeier), lakmal.seneviratne@kcl.ac.uk (L. Seneviratne).

0925-2312/01/$ - see front matter
PII: S0925-2312(00)00307-6

a moment of hesitation. Developing a robot with the same kind of fluency in its dealing with the environment is a tremendously challenging task.

Unlike many other tasks that seem trivial to human beings but are beyond the capabilities of current technology, navigation tasks, such as goal reaching and obstacle avoidance, are tasks for which specific rules can be written down. We claim that using this feature is beneficial to solve these tasks effectively. The fact that rules can be written down for the accomplishment of a task leads naturally to a rule-based system with which to tackle these tasks. In contrast to the black box nature of neural-network-based systems, a system based on rules has the advantage of transparency and interpretability. Furthermore, prior knowledge about the tasks can be incorporated into the rule base whereas such an initialization is cumbersome in a neural network. Prior knowledge of a problem domain is, however, seldom enough to build a system that can effectively deal with idiosyncratic requirements or changes in an environment. Neural networks have been used in navigation and control applications to deal with such problems. Powerful learning algorithms based on theories of neurobiological control [2] have been devised that allow neural nets to change their behavior in response to a changing environment. In the method for navigation we propose in this paper, we combine neural and fuzzy techniques. This introduces the flexibility and adaptability of neural networks into the more rigid rule-based fuzzy system that captures the basic, unchanging, aspects of the problem domain. We believe that this hybrid neuro-fuzzy approach to control is a fertile direction for research into transparent yet adaptive control systems.

Fuzzy logic was invented to reduce and explain system complexity. Zadeh was concerned with the rapid decline of information extractable by traditional mathematical models with increasing system complexity [22]. Much of this complexity stems from the way in which the variables of a system are represented and manipulated. Since traditional variables could only represent the state of a phenomenon as either existing or not existing, the mathematics necessary to evaluate operations at boundary states or gray zones become increasingly complex. In fuzzy logic this problem is tackled by reasoning in linguistic rather than numerical terms. This reduces the complexity of the description and can thereby increase the level of understanding. For engineering purposes, the use of fuzzy logic simplifies the communication between experts that know the problem domain and experts that build a control system to cope with the problem domain. Moreover, a fuzzy linguistic representation usually is a more accurate reflection of the knowledge a control system has about its noisy environment.

The design goals we aim for in this paper are threefold. First, we aim to develop a system that can deal with the real-time requirements of robotic manipulation. Such speed is usually gained at the cost of the inability to find a solution in every circumstance. This problem of local minima is a serious one and we will not solve it in any generality. We will, however, suggest some possible approaches in the discussion. The related second goal is to develop a system that scales well with the addition of further degrees of freedom. Local calculation of solutions and a minimum of communicational overhead are required for this. Thirdly, we aim for a transparent design. Being able to assign a specific role to the various parts of the navigator not only allows

a human engineer to understand and adapt the system, it also allows a learning rule, based on neural reinforcement learning, to operate and generate a working system.

In Section 2, we discuss how a navigation system for robotic manipulators can be based on fuzzy logic. The intuitive prior knowledge about the domain of goal reaching and obstacle avoidance as well as some trial and error procedures are used to develop a rule base that can be used to steer the MA2000, an experimental manipulator. The results in Section 4.1 show that the non-adaptive, rule-based system copes effectively with the kinematic requirements of robot navigation. The navigator has been tailored to carry out a particular task, but the controller generalizes well as long as the shape of the manipulator and the surrounding environment stay within certain bounds.

However, greater changes in the environment or in the manipulator itself can drastically cut down the performance of a navigation scheme. Humans in such a situation (for example those whose limbs become permanently or temporarily impaired) will quickly adapt their control system to cope with the changed "manipulator". Also, humans can adapt to a change in environment with ease. In Section 3, we show how the navigator can be made adaptive by adding a mechanism based on the reinforcement learning paradigm [3]. This neural network learning mechanism allows the rule base, or rule *network*, to adapt to new environments as well as to changes in the requirements or to physical restrictions of the manipulator. Simulations using the adaptive navigator are presented in Section 4.2.

1.1. Related work

In developing our navigator, we were first of all influenced by Khatib's obstacle avoidance method which is based on an *artificial potential field* [8]. Khatib computes an artificial potential field that has a strong repelling force in the vicinity of obstacles and an attracting force produced by the target location. The superposition of the two forces creates a potential field which incorporates information about the environment. Following the steepest gradient from a start position, a path is usually found that guides the robot to the target position while avoiding obstacles [11]. In our approach, the amount of computation that is required is reduced by using only the nearest obstacles to determine the direction of motion, while in most implementations based on Khatib's potential field method every obstacle is utilized for the computation of a new force vector. This restriction is a choice for speed rather than completeness and is determined by our design goals.

Khatib's method may fail to find an existing path to the goal. This problem which is inherent to all local navigation methods is called the *local-minima* problem [11]. A possible solution for this problem is suggested in Section 5.

Secondly, the system described here is also related to the subsumption architecture developed by Brooks [4]. The main principles of his work are a collection of modules which are interconnected on different layers with different hierarchies. These modules are for example wall following, obstacle avoidance, goal reaching, etc. Depending on sensory input, a module becomes active and generates a command for the robot [4]. While Brooks' system resembles an expert system where for any input signal one specific reaction module or a specific combination of modules is active, the fuzzy

approach is a parallel processing approach and each input contributes to the final decision.

Thirdly, our navigator can be seen as a behavior-based or reactive system which possesses knowledge — implemented by a human designer or/and induced by a training algorithm — on how to react by using information about the nearby environment [20,13]. Such a robot navigator is confronted with a variety of tasks and a large number of sometimes conflicting soft as well as hard constraints. The designer of the rule base or the learning algorithm has to tune the system such that it integrates the different aspects of the navigation problem at hand in the best possible way [13]. Due to the transparent nature of the control system, this tuning is simplified in the approach we describe here. Tuning, be it of the rule base or the learning algorithm, nevertheless remains an important aspect of the development.

Finally, we will be discussing a reinforcement learning mechanism. This is different from the "traditional" reinforcement learning as applied to mobile robots [19,5]. In those approaches, the robot is represented by a point in a grid. The robot moves in an obstacle-cluttered environment and is rewarded for finding the goal. With time the robot builds up a map of its environment which it can use for path planning. Eventually, this map will give a complete description of the environment. These map-building approaches commonly suffer from the long time spent on exploration and tend not to be effective in dealing with dynamic environments. In contrast to this approach, our method is fast and, thus, suitable for real-time navigation. On the negative side, our method may get stuck in a local minimum and may occasionally be unsuccessful in finding the desired goal configuration. See Section 5 for a further discussion.

The use of fuzzy-based systems, similar to our reactive system, has recently attracted a great deal of interest in those research communities which focus on the development of autonomously guided vehicles and intelligent mobile robots [6,20,13,9,15–17]. Those fuzzy-based navigators provide a mobile robot with an intelligent behavior in reaction to the nearby environment by combining the competing functions of goal following and obstacle avoidance.

Although the focus of this paper is on the simulation of the neuro-fuzzy navigator and an investigation of its learning capabilities, we have shown in recent papers that the fuzzy navigation principle can also be applied to real-time manipulator systems [1,2].

2. The non-adaptive navigator

Non-branching manipulators with revolute joints are widely used in industrial applications. Even though these robots are still far from the complexity of the human arm, they posit a non-trivial navigation problem. When developing a particular navigation method it is important to realize that the method must be generally applicable to have any real validity. We have developed our navigator with manipulators such as the PUMA-type and SCARA-type manipulators in mind to be able to test our algorithms in a realistic environment. The manipulator investigated in our

experiments is the MA2000 manipulator which has four degrees of freedom in its current configuration. This manipulator has kinematic properties similar to PUMA-type robot arms. Since the work described in this paper focuses on planar motions, our results can be applied to PUMA-type as well as SCARA-type robot arms [11,21].

One of our aims is to develop a mechanism which allows navigation in real time. This constraint would seem to rule out global path-planning algorithms that incorporate into their decision process the position of all obstacles and all manipulator links [11]. Such an approach would almost certainly fail to meet the real-time requirement for multi-joint manipulators in densely populated environments. To lower the cost of scaling-up the number of joints as well as the number of obstacles, our navigation method is based on local information. Firstly, each link movement is controlled by a separate, local, navigation unit, which is mainly influenced by the position of obstacles in its immediate vicinity and the goal configuration. Secondly, any exchange of information between different link units is kept to a minimum.

Each of the navigation units, which individually control a link, receives two main inputs. Firstly, the distance between the link and the nearest obstacle, and secondly, the angular difference between the current link configuration and the target configuration. The output variable of such a navigation unit is a motor command which is sent to the joint motor of the corresponding manipulator link. All variables take on positive as well as negative values and therefore provide information about the magnitude as well as the sign of displacement relative to the link — left or right. The motor command is fed to the motor at each link move (Fig. 1 and [1]). To calculate the distance to an obstacle, only those obstacles that fall into a bounded area surrounding each link are considered. In the current implementation these *scanning areas* are chosen to be of rectangular size (Fig. 1). The rectangular scanning areas

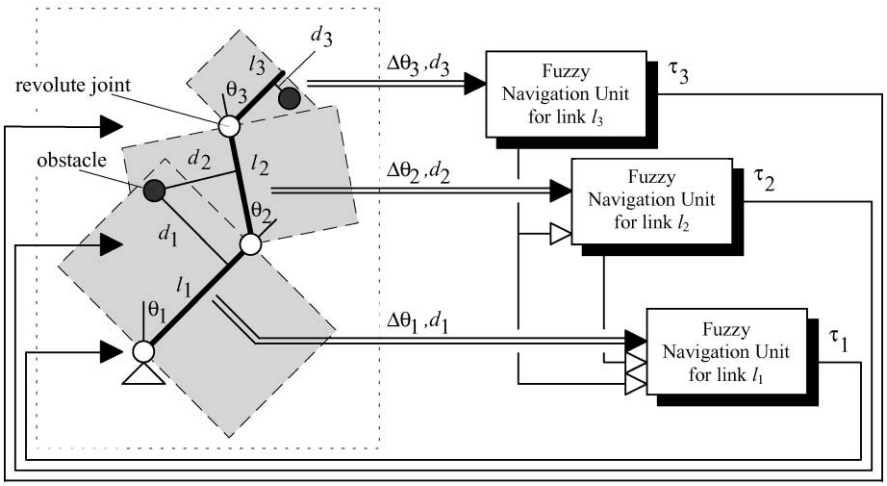

Fig. 1. A 3-link planar robotic manipulator connected to fuzzy navigation units. The shaded areas depict the obstacle scanning areas. If no obstacle is detected inside the scan area, the navigation unit is informed of an obstacle in the far distance.

are a model of the space scanned by ultra-sonic sensors attached to both sides of a link [14].

A unit has to produce an appropriate motor command from the sometimes contradictory inputs. On the one hand the unit has to lead the corresponding link to its goal position, on the other hand it has to force the link to back-up when approaching an obstacle. From our intuition or common sense knowledge, we can write down rules that capture this decision logic well:

- If an obstacle is right and the goal is close left then make a small move to the left,
- If an obstacle is very close left and the goal is left then make a big move to the right.

Using intuition and expert knowledge, it is not difficult to write down a multitude of such rules describing the different manipulator scenarios. The power of fuzzy logic is that it can translate these intuitive rules into constructs for the development of control systems [17]. During the development of the proposed navigation system, maximum possible use of prior knowledge of the problem domain can be made. Although the interaction of a "multitude of rules" produces effects that are not represented in any rule alone (emergent phenomena), the individual rules give a good indication on how the overall system functions and a good explanation on why the manipulator moved the way it did. This is in stark contrast to highly distributed control systems such as neural networks.

The local units operating on their own are not very likely to come up with the ideal solution to the navigation problem at hand. Some form of communication between the units is required for navigation through a complex environment. We investigate two approaches with respect to the inter-link communication. In [1,2], a system was introduced in which the units of proximal manipulator links (links which are close to the base) have additional inputs which are connected to the fuzzy outputs of more distal links (links which are distant to the manipulator's base). In Fig. 1, the inter-link communication is shown by lines that connect units of distal links with those units corresponding to proximal links. Using these extra connections, the movement of the proximal links also depends on the immediate environment of the distal links. For example, a close-by obstacle not only makes the particular link dodge, but also forces the more proximal links to join the evading maneuver. This approach provides robust performance, especially regarding the avoidance of collisions between obstacles and distal links despite fast progression of more proximal links. However, the number of rules to be set up for the unit of proximal links can be large and difficult to manage. This approach has been used for the non-adaptive navigator [1,2].

A different inter-link communication has been adopted in conjunction with the adaptive navigator. In this approach, the number of inputs remains two for any link unit. The first input describes — as before — the difference between current configuration and goal configuration. As second input, the ith unit receives "min(distance(l_i), distance(l_{i+1}), … , distance(l_n))", where n is the number of links. As an example, consider a two-link manipulator: whenever the distance between any one obstacle and link l_2 is smaller than the distance between any one obstacle and link l_1, the navigation unit of l_1 will also be influenced. Compared to the above approach, this

approach further reduces the communication overhead, reduces the number of rules for the navigation units of proximal links, and proved to be very robust.

2.1. The fuzzy algorithm

This section describes how common sense knowledge about obstacle avoidance can be translated into the formal language of fuzzy logic. The rules such as the two stated earlier in this section can be written as sentences with two conditionals and one conclusion. This structure lends itself to a tabular representation as shown in Table 1. This table represents our prior knowledge of the problem domain. Notwithstanding the practical difficulties of translating this knowledge into a control system, we consider it to be important to use this prior information. In a pure neural network-based approach such prior knowledge would be difficult to encode.

In fuzzy logic terms, the range of a variable is called the universe of discourse. To translate Table 1 into fuzzy logic, the universe of discourse D that describes obstacle distance $d \in D$ is partitioned by fuzzy sets $A_1(d), \ldots, A_p(d)$, where p is the number of fuzzy sets. Each set $A_k(d), k = 1, \ldots, p$, represents a mapping $A_k(d)$: $D \to [0,1]$ by which d is associated with a number in the interval [0,1] indicating to what degree d is a member of the fuzzy set. Since d is a measure of distance, "*close_left*", for example, may be considered as a particular fuzzy value of the variable distance and any d is assigned a number $A_{close\_left}(d) \in [0,1]$ which indicates the extent to which that d is considered to be *close_left* [12,9]. Similarly, fuzzy sets $B_m(\Delta\theta), m = 1, \ldots, q$ can be defined over the universe of discourse Θ that represents the angular distance to the goal, $\theta - \theta_{target} = \Delta\theta \in \Theta$. In contrast to the Mamdani controller, our Sugeno-type controller [10,17,18] has an output set that is not partitioned into fuzzy sets (Fig. 2). Thus, the rule conclusions consist of scalar actuator values C_j, $j = 1, \ldots, r$, where r is the number of the rules.

A variety of functions can be employed to represent fuzzy sets. Here, we use asymmetrical triangular and trapezoidal functions to describe the fuzzy sets $A_k(d)$. This choice is based on the fact that these fuzzy sets allow the fast computation that is

Table 1
A rule base for a single link of a manipulator. This rule base is a translation of the common-sense knowledge of the problem domain into the language of fuzzy logic. Columns ($A(d)$) represent the fuzzy measures of the distance to an obstacle, while rows ($B(\Delta\theta)$) are fuzzy representations of the distance to the goal. Each element of the table can be interpreted as a particular motor actuation command

$B(\Delta\theta)$	$A(d)$					
	far left	left	close left	close right	right	far right
far left	left small	right big	right very big	left very big	left big	left big
close left	left small	right very small	right big	left big	left big	left small
contact	nil	nil	right small	left small	nil	nil
close right	right small	right big	right big	left big	left very small	right small
far right	right big	right big	right very big	left very big	left big	right small

essential under real-time conditions. Defining the parameters, *ml* and *mr* as the *x*-co-ordinates of the left and right zero crossing of the trapezoids, and *mcl* and *mcr* as the *x*-co-ordinates of the left and right side of the trapezoid's plateau, the trapezoidal functions can be written as

$$A_k(d) = \begin{cases} \max\left(\dfrac{d - ml_k}{mcl_k - ml_k}, 0\right) & \text{if } d < mcl_k \\ 1 & \text{if } mcl_k \le d \le mcr_k, \qquad k = 2, \ldots, p-1, \\ \max\left(\dfrac{d - mr_k}{mcr_k - mr_k}, 0\right) & \text{if } d > mcr_k. \end{cases} \tag{1}$$

Triangular functions can be achieved by setting $mcl = mcr$. At the left and right side of the interval the functions are continued as constant values of magnitude one

$$A_1(d) = \begin{cases} 1 & \text{if } d \le mcr_1, \\ \max\left(\dfrac{d - mr_1}{mcr_1 - mr_1}, 0\right) & \text{if } d > mcr_1 \end{cases} \tag{2}$$

and

$$A_p(d) = \begin{cases} \max\left(\dfrac{d - ml_p}{mcl_p - ml_p}, 0\right) & \text{if } d \le mcl_p, \\ 1 & \text{if } d > mcl_p. \end{cases} \tag{3}$$

The fuzzy sets, $B_m(\Delta\theta)$, to fuzzify $\Delta\theta$ are defined analogously. Fig. 2 shows the fuzzy sets $A_1(d), \ldots, A_p(d)$ and $B_1(\Delta\theta), \ldots, B_q(\Delta\theta)$.

Each of the fuzzy sets, A_k and B_m, are associated with linguistic terms ALT_k and BLT_m, respectively. Thus, the linguistic control rules $R_1, \ldots, R_r$, which constitute the fuzzy rule base, can be defined as

$$R_j: \quad \textbf{IF } d \textbf{ is } ALT_k \textbf{ AND } \Delta\theta \textbf{ is } BLT_m \quad \textbf{THEN } CLT_j, \tag{4}$$

where CLT_j is a linguistic term associated with the actuator values C_j, $j = 1, \ldots, r$ and r is the number of rules.

The *t*-norm product operator gives a mathematical interpretation of the **AND** operation:

$$G_j = A_k(d) \cap B_m(\Delta\theta) \underset{\left(\begin{smallmatrix}\text{using the} \\ \text{product } t\text{-norm}\end{smallmatrix}\right)}{=} A_k(d) * B_m(\Delta\theta). \tag{5}$$

Finally, the output of a fuzzy unit is given by a weighted average over all rules (Fig. 2 and [10]):

$$T = \frac{\sum_{j=1}^{r} G_j \cdot C_j}{\sum_{j=1}^{r} G_j}. \tag{6}$$

Eq. (4) together with Eqs. (5) and (6) define how to translate the intuitive knowledge reflected in Table 1 into a fuzzy rule base. The details of this translation can be

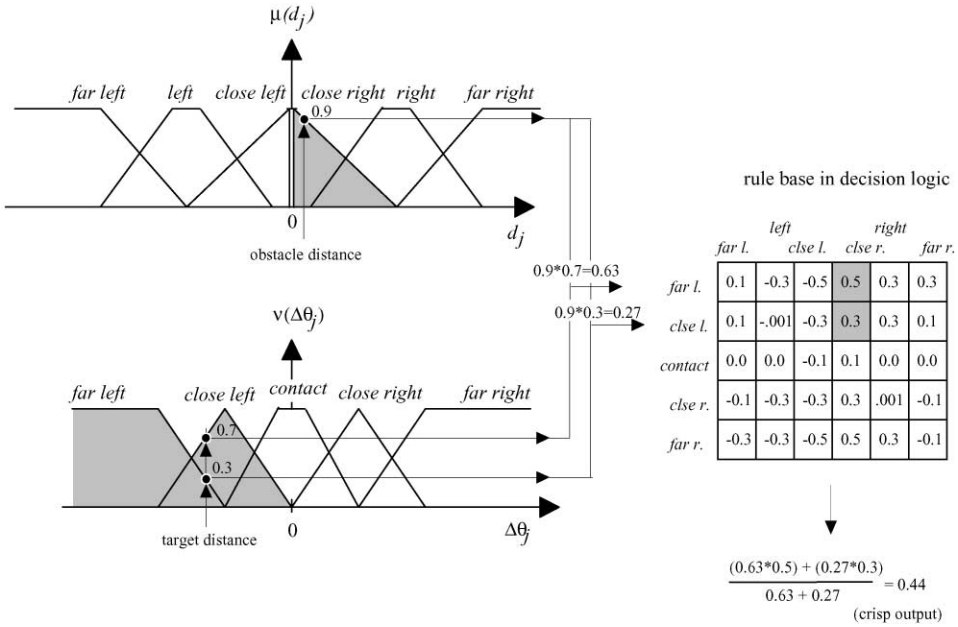

Fig. 2. Schematic depiction of fuzzification stage, decision logic and defuzzification of a fuzzy controller used for the robotic navigation task. The two inputs ("difference between actual and target position" (d) and "distance to obstacle position" (θ)) are partitioned into a number of fuzzy sets. Two crisp input values are indicated by "obstacle distance" and "target distance". These input values activate the corresponding fuzzy sets (shaded fuzzy triangles) which are then combined using the fuzzy product operator. In the next step, two actions in the two-dimensional rule base (shaded squares) are activated. These two actions are combined (weighted sum) to produce one crisp output. The crisp output represents an actuator command which lies between "right very big" (0.5) and "right big" (0.3).

modified by changing the number of fuzzy sets, the shape of the sets (by choosing the parameters, ml, mr, mcl and mcr) as well as actuator value C_j of each of the rules in Eq. (6).

As an example, a scheme that implements the knowledge embedded in Table 1 as a fuzzy navigation unit is shown in Fig. 2. In this particular application, the number of fuzzy sets which fuzzify the obstacle distance, d, and the distance to the goal, $\Delta\theta$, are chosen to be six and five, respectively. All other parameters have been refined either by trial and error (Section 2.1) or by the reinforcement learning algorithm (Section 3.1).

3. The adaptive navigator

There are several reasons why the control system as it stands is not quite satisfactory. Most importantly, even though a rule base developed on the basis of intuition leads to a reasonable performance in a simulated environment, mistakes would be costly in the real world. The work in Section 4.1 solves this problem by modifying the

rule base on a trial and error basis. Such an approach, however, soon leads to intolerably long development times.

The second issue is one of flexibility. The rule base will be different for different robot arms and there is no simple relationship between the rules that are effective for one robot or another. For instance, navigators for a long and narrow or a wide and short link are not simply related by a scaling of the rule base due to the conflicting requirements of goal finding and obstacle avoidance. This issue becomes even more important when it is taken into account that many of these manipulators carry different loads at different times. Such changes can significantly affect the kinematics of a robot (considering a load as an extension of the most distal manipulator link) and would require changes in the rule base.[1] Clearly, the development of a new rule base for each new situation is not desirable.

A third issue concerns optimization of motion. Slow motion may be sufficient or even required in some environments, but in sparsely populated environments the link might as well take larger steps. This would lead to faster goal finding which in most situations will be seen as good performance. Faster movement, however, requires a change in the rule base. Flexibility in the rule base is therefore also required to deal with varying environments in an optimal way.

Within the fuzzy rule-based framework these issues are difficult to deal with. Neural network models, on the other hand, allow a control system to learn from its own mistakes by evaluating its performance against a set of criteria and adjusting the implicit knowledge of the (current) problem domain. The adaptive algorithm we present is based on an actor-critic model [3]. These models divide an adaptive controller in two parts. The first is the actor, which performs actions in response to input from the environment. Depending on the changes in the environment after an action, the critic will send a reward (or punishment) signal to the actor to change its rule base such that the action will be more (or less) likely in the future. Here, the actor is the original, non-adaptive navigator and the critic is the new mechanism that adapts the rule base. The critic receives information from the sensors and can be loaded with different rules that determine whether a sensory input represents a desirable situation or not.

3.1. The neural learning algorithm

This section gives the formal description of the adaptation of the rule base. The training algorithm described here applies to the rule base only (Table 1). By adding this learning stage, the rule base becomes a more flexible structure that we refer to as the rule *network*. The rule network is modified every time the corresponding robot link moves due to an actuator command from the navigation unit. This motion is referred to as a training step in the following. The fuzzification stage (parameters such as number, width and position of the trapezoidal fuzzy sets (see also Section 2.1)) is not

[1] Note, picking up a load by a manipulator usually also changes the system's dynamics. However, our focus is on a purely kinematic system. The treatment of dynamic aspects is beyond the scope of this paper.

the target of the training method we propose here (see Section 5 for a further discussion).

The actor in the adaptive controller is identical to the controller of the navigator discussed in Sections 2 and 2.1. The navigator determines its actions on the basis of the distance to the nearest obstacle d and the target $\Delta\theta$. The critic can determine whether to punish or reward an action by inspecting the manipulator's performance. At each training step, only rules, which are actively involved in contributing to the decision process, are adapted by a reinforcement command, while rules, which do not contribute to the particular movement, remain unchanged. Four performance criteria are used for training: (1) distance between the link and obstacles, (2) the speed of the link, (3) the goal reaching performance and (4) the angle between link l_2 and link l_1:

(1) Whenever the distance between link and obstacle falls below a given threshold, the values of those active (= responsible) rules which proposed a movement towards the obstacle are decreased by a small amount (punishment), while those active rules which proposed a movement away from the obstacle are increased (reward). Once the navigator is trained and keeps the manipulator at sufficient distance to obstacles, the training command automatically becomes inactive.
(2) The training with respect to the manipulator's speed is split into two parts: the navigator is trained to avoid both very large and very small changes in the position of the manipulator. The first of these two training commands is useful to avoid big sudden jerks of the manipulator link. The second is active when the link barely moves at all. Again, active rules which support the desired performance are increased, those which are opposing it are decreased. Both commands are inactive when the manipulator performs correctly. A moderate speed is the desired aim of this training part.
(3) The navigation unit is trained to improve the goal reaching performance by increasing the weights of those active rules which make the manipulator link move towards its final position. Active rules which do not support this are suppressed. This command becomes active only when the goal position cannot be reached within a certain number of steps.
(4) An unlimited rotation of link l_2 around link l_1 is not desired, since the motion of the links of the MA 2000 is limited to a range of $\pm 0.75\pi$. Thus, whenever the link l_2 tends to "collide" with link l_1 the rule network is appropriately altered. Again, an action which favors a motion in the expected direction (towards a stretched configuration of link l_2) results in the increase of the value, while an action which suggests the opposite motion (towards a collision with link l_1) is decreased. This training command is active only if the angle of link l_2 is greater than 0.64π or smaller than -0.64π.

The whole set of all four training commands is applied every time the arm moves by a single training step. If after 100 training steps the goal has not been reached, the training is interrupted, the manipulator put back into a start position, and a new training cycle is started. A training cycle is completed, once all links have reached their final configuration or 100 training steps have been carried out. In order to assure that

the unit is able to deal with obstacles on the right and left after training is completed, start and goal configuration were exchanged every second training cycle.

4. Results

This section describes the conducted experiments and achieved results using the non-adaptive and adaptive navigation strategies. In the first part, Section 4.1, the fuzzy navigation method has been used to steer a model of the MA2000 manipulator, while in Section 4.2 the proposed learning method has been added.

4.1. Results for the non-adaptive navigator

The non-adaptive navigation method has been applied to two as well as three links of the MA2000 manipulator. The results are shown in the form of snapshots of the manipulator's motion through an example environment. The figures show the position of the obstacles and robot arm in the two-dimensional workspace at various times during its movement from start to goal position (Figs. 3 and 4). As only links with zero inertia are considered in these experiments, the applied motion command is an instantaneous angular step, which modifies the previous velocity without time delay. Recently, this method has been successfully applied to a physical manipulator showing that our navigation method can also deal with the dynamic aspects of manipulator navigation [1,2].

Fig. 3 shows the navigation of the MA2000 employing the non-adaptive navigator. In Fig. 3 (left), the navigator was used to steer two links of the MA2000. The continual tendency of link l_2 to reach its goal configuration and its slowing down and backing-up near obstacles is obvious. In Fig. 3 (right), the navigator has been applied to the three-link MA2000. After developing the rule base we tested the navigators in a number of environments with obstacles of different shape, size and position. The controller found obstacle free paths in almost all environments tested as long as the

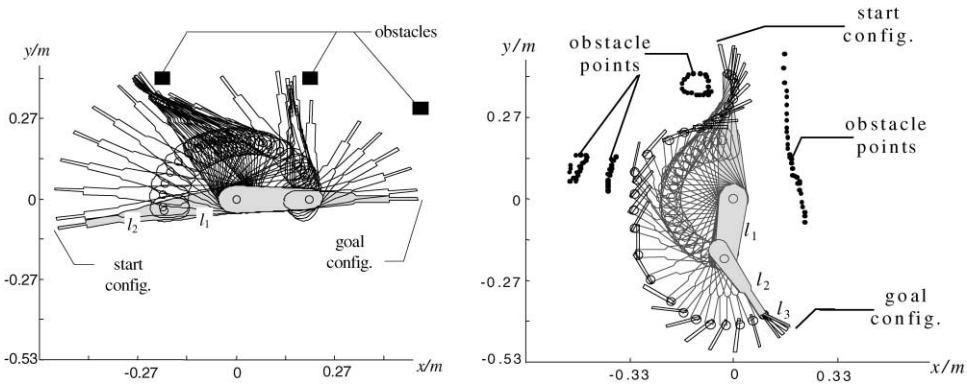

Fig. 3. Planar motion of the MA2000 manipulator using the non-adaptive navigator. Left: two links. Right: three links.

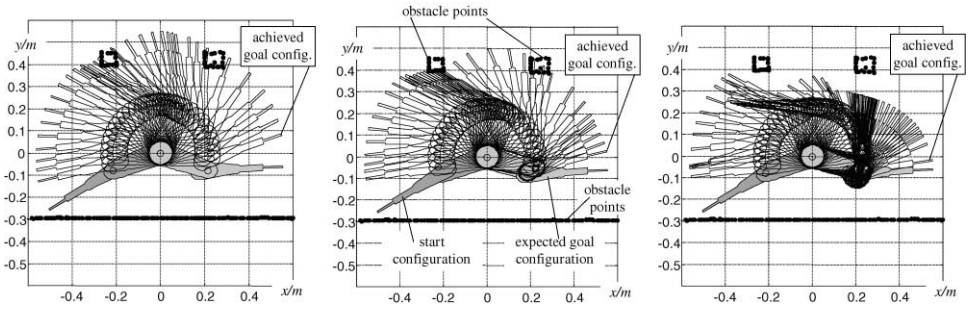

Fig. 4. Training of the navigation unit of link l_2. The training is shown at different training cycles. Left and center: After a few training cycles, the link begins to react to obstacles, but still crashes into them. Right: After about 10 cycles the initially untrained rule network stabilized. Goal tracking improves continuously. Further training changes the performance of the manipulator only marginally and the rule network experiences only minor changes. The center figure shows the goal, start and obstacle positions for all three figures.

positioning of the obstacles allowed for a collision free path [2]. In some rare cases collisions occurred. Those occurred when the density of the objects in space was increased much beyond the density used for the development of the rule base. This can be understood as the effect of an average speed that the system has settled down to, which is too high for densely populated environments. Reducing the magnitude of the actuator commands would solve this problem for a particular environment but as a general solution this would not seem to be adequate. However, in nearly all of those experiments where the system was trapped in a local minimum, and the manipulator was incapable of reaching the goal configuration, the system avoided collisions with nearby obstacles [1]. Thus, a well-adjusted fuzzy navigator gives higher priority to obstacle avoidance than to goal tracking — this is highly desirable considering the safety of robot and environment.

In a further test of the navigator, we tested the sensitivity to changes in the rule base. In a real system such changes could be due to noise in the system or imperfections of the translation of the actuator commands into steps of the arm. We found that the navigation was robust against some changes in the rule base but especially the values in the center of the rule table (Table 1) had to be chosen quite carefully. Such careful tuning naturally leads to a long development process which has to be repeated after changes in the link structure, or large increases in the density of the obstacles in the environment. We think these problems first of all show the limits of using a single, fixed rule base and secondly, the limits of our own intuition about the domain. The results presented in the next section show that a navigator that employs a learning algorithm can cope with these problems.

4.2. Results for the navigator trained by the neural learning algorithm

We tested the learning algorithm on link l_2 of the two-link MA2000. The rule network of the corresponding navigation unit was initially set to zero and the arm was

placed in an environment as shown in Fig. 4. Initially, the manipulator does not move at all, because all values in the rule network are set to zero. Owing to the influence of training command 2, the manipulator starts to move after a few training steps. In this phase the manipulator collides with the obstacles and fails to find the goal (Fig. 4 (left and center)). Now however, training commands 1 and 3 begin to influence the learning process. The rule network is changed such that towards the end of the learning process, the manipulator will have found a collision-free path from the start configuration to the goal configuration or a configuration nearby (Fig. 4 (right)).

To investigate the performance of the navigator in more detail, we monitored several aspects of its behavior during learning. Fig. 5 (left) shows the average distance between link l_2 and the nearest obstacle that was kept during training. In the beginning of the training, the average distance was very small (accounting for many collisions during the training cycle), but the punishments received from the critic for this misbehavior altered the rule network and soon led to paths in which the obstacles were avoided. The rule network settled down to a state in which the distance to the obstacles was kept slightly above 0.4 m. This particular value depends on parameters such as the density of the distribution of the obstacles, the location of the goal position and the various training parameters. Changing these learning parameters allows one to tune the performance of the navigator to the idiosyncratic requirements of the environment or task at hand. For instance, when safety is a premium requirement, the parameters can be chosen to increase the distance that is kept from obstacles.

While training to avoid obstacles, the navigator also attempted to make large steps to reach its goal more quickly. Fig. 5 (right) shows that while the initial steps are small, the navigator reaches a step size of 0.15 rad round about the same time as the network learned how to avoid obstacles. Finally, the goal searching performance was monitored. Whereas the initial rule network led to large oscillations around the goal, the learning command adapted the rule network to stay close to the goal. Summarizing the three performance measures, the navigator starts out not knowing where the goal is, moving too slowly and bumping into obstacles, but converges to a state in which it rapidly moves towards the target location along a collision free path.

The transparency of the control system allows us to monitor the evolution of the rule base during the learning process. Table 2 shows the numerical values of the rule base after 150 iterations when the rule base has settled into a stable state (see Fig. 5, bottom). The corresponding linguistic rules, which are derived using the conversion table (Table 3), are presented in Table 4. This type of representation allows an interpretation of the learned behavioral rules. The generated rule base shown in Table 4 looks different from the one shown in Table 1, which has been constructed by hand. However, a close examination reveals that these two rule bases provide a very similar behavior.

For the following discussion, we focus on the left-hand side of rule base tables. This can be done without loss of generality, since the rule base has a mirror-symmetry around the center of the table. For instance, the value "0.7772" appears twice in Table 2; firstly, if an obstacle is close left and the goal is far left, and, secondly, if an obstacle is close right and the goal is far right. The only difference between the two rule values is their sign. Note that this mirror symmetry is built into the learning

Table 2
The rule base for link l_2 of the MA2000 manipulator after 150 iterations of training using the reinforcement learning algorithm. The scalar value in each cell of the table can be interpreted as a particular motor actuation command

$B(\Delta\theta)$	$A(d)$					
	far left	left	close left	far right	right	close right
far left	0.0739	0.3754	− 0.7772	0.1931	0.0635	0.0774
close left	− 0.0008	0.3977	− 0.8018	0.1514	0.0247	− 0.0004
contact	− 0.0000	− 0.1218	− 0.4242	0.4242	0.1218	0.0000
close right	0.0004	− 0.0247	− 0.1514	0.8018	− 0.3977	0.0008
far right	− 0.0774	− 0.0635	− 0.1931	0.7772	− 0.3754	− 0.0739

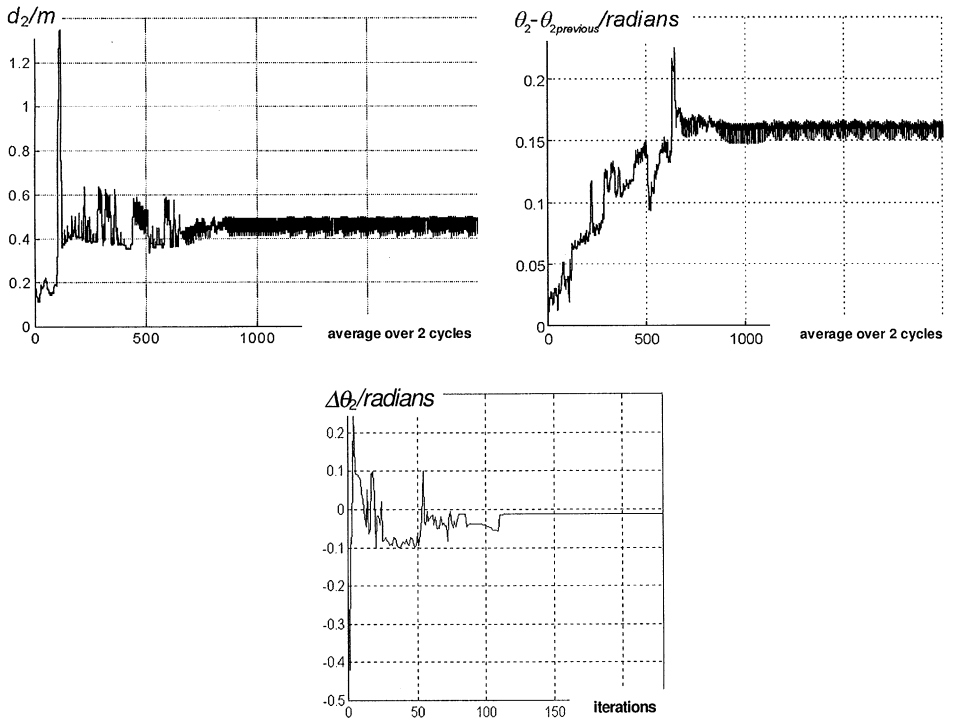

Fig. 5. Left: Average distance between link l_2 and the nearest obstacle. Right: Step width of link l_2 in radians. Bottom: Difference between final configuration and desired configuration for each iteration reaching the goal from the left. Similar results were recorded for the opposite direction. The performance measures in the left and the center figure are shown as running averages over 2 training cycles.

algorithm: whenever a rule is changed by a certain amount during training, the "mirrored" rule is changed by the same magnitude but with an inverse sign.

Comparing Table 1 with Table 4 shows that if an obstacle is close left (independent of the position and distance of the goal), the rules suggest a movement to the right.

Table 3
Conversion of numerical ranges into linguistic rules

Rule	Numerical range
right very big	$[-1 \ldots -0.5)$
right big	$[-0.5 \ldots -0.2)$
right small	$[-0.2 \ldots -0.001)$
right very small	$[-0.001 \ldots 0)$
nil	0
left very small	$(0 \ldots 0.001]$
left small	$(0.001 \ldots 0.2]$
left big	$(0.2 \ldots 0.5]$
left very big	$(0.5 \ldots 1]$

Table 4
The linguistic interpretation of the rule base shown in Table 2

$B(\Delta\theta)$	$A(d)$					
	far left	left	close left	far right	right	close right
far left	left small	left big	right very big	left small	left small	left small
close left	right very small	left big	right very big	left small	left big	right very small
contact	nil	right small	right big	left big	left small	nil
close right	left very small	right big	right small	left very big	right big	left very small
far right	right small	right small	right small	left very big	right big	right small

The difference is in the magnitude. The rule base of Table 1 suggests "right very big" movements when the goal is far away and decreasingly smaller right movements the closer the link comes to the goal. These commands have been established using common sense and several test runs. In contrast, the rule base of Table 4 suggests "right very big" movements if the goal is on the left and decreasingly smaller right movements for contact and goal on the right. The "close left" columns of both rule bases make sense. The particular rules generated by the learning algorithm depend on the specific obstacle constellation seen by the adaptive navigator during learning.

Although there are differences between the "left" and "far left" columns of the two rule bases, respectively, a common trend can be observed. The two "close right" and "far right" rows are similar as far as they mainly contain rules commanding right movements. There is only one exception in Table 4 where a "left very small" command appears in the "far left" column instead of a "right small" command in Table 1.

The greatest difference can be observed in the "far left" and "close left" rows. Especially, the two "left big" commands in the "left" column of the rule base generated by the learning algorithm (Table 4) differ quite strongly from the "right" commands in the hand-made rule base (Table 1). However, even here a trend can be recognized. While the hand-made rule base suggests a more gradual decrease for the commands over all three columns ("close left", "left", "far left"), the automatically generated rule base changes abruptly from "right very big" in the "close left" column to "left big" in the "left" column. The "right very small" command in the cell of the "far left" column and the "close left" row in Table 4 may have been generated in an attempt to compensate the "big left" commands in the "left" column.

In conclusion, the rule base generated by the learning algorithm can be interpreted in linguistic terms, and this allows an intuitive comparison of different rule bases. Describing the control system in linguistic terms makes it easier to discover trends in the strategy that the navigator pursues to cope with the requirements of its environment. Discovering such trends can be useful to fine tune the rule base, to spot errors in the training procedure, or to develop improved training strategies.

5. Discussion and conclusion

Our main aim in this contribution was to present a hybrid neural and rule-based strategy which we have found to be successful for the manipulator navigation task. Section 2 showed how elements of this strategy are used to develop a neuro-fuzzy navigator for robotic manipulators. This approach, which implements behavior as a set of linguistic rules, is intuitive and easily allows for the addition of further layers of control and fine-tuning. In this section we discuss some of the further possibilities as well as pitfalls.

It is important to stress again that the method developed here is a local method. Global methods that consider the whole obstacle space at each iteration and compute an optimal obstacle-avoiding path to the goal will always find a collision-free path if one exists. In a real environment, however, the costs of calculating global trajectories are prohibitive. Systems that rely on this are unlikely to achieve real-time performance in complex or non-stationary environments [11]. Rather than using a method which will turn out to scale poorly when moving to more complex environments or manipulators, we have concentrated on finding a fast local method that can perform in real time.

We realize that this choice leads to the problem of local minima: there may be routes leading to the goal that avoid the obstacles which our method cannot find. Even though this is so in theory, our experiments have shown that our method can find a solution in many cases. The problem of local minima, however, cannot be ignored. It is possible to create an environment in which a manipulator controlled by our method will never reach the goal [1]. A first option to circumvent such problems is to use more information from the environment than we currently do. Rather than looking at the nearest obstacles only, a larger number of obstacles could be considered in the decision process. This would increase the number of environments the system

can deal with at the cost of extra computations that have to be performed at each time step. In line with the adaptive mechanism developed in Section 3, we are currently considering to develop a control mechanism that uses an adaptive scanning range. This mechanism should be able to base its decision process on varying numbers of obstacles depending on the situation it finds itself in. Such a mechanism, when successful, would strike a compromise between the necessity of expensive global computations in one environment and the cheap and fast local processing that is possible elsewhere.

Our experiments with the learning algorithm were promising. The learning algorithm was able to create a useful rule base. Especially, the obstacle avoidance problem was dealt with in a satisfactory way. We believe that the success of the learning algorithm is due in great part to the transparent nature of the control system. This transparency not only allowed us to directly implement prior knowledge about the problem domain in Section 4.1, but it also reduced the credit assignment problem in such a way that the learning algorithm could quickly adapt the rule base from tabula rasa to a functioning navigator. Neural network methods represent rules in a highly distributed manner, which makes it difficult to assign a specific role to a particular node in the network. This leads to the credit-assignment problem: as it is unclear what the role of a node is, it is unclear whether it should be punished or rewarded when a particular situation occurs. Neural networks commonly solve this problem by using error back propagation, which is a notoriously slow process. In the current approach, however, all nodes (all rules) are easily interpretable in every day terms and even the learning commands can be stated in every day terms. This speeds up the learning and leads to the fast convergence times observed in the experiments.

Slight problems were encountered with the goal tracking. The navigator whose rule base was created from scratch by the learning algorithm was able to direct the manipulator in the right direction. Although the manipulator reached the area near the goal configuration, the created rule base did not allow accurate docking maneuvers. A possible solution to this problem is to use an additional module which takes over to move the manipulator into its desired configuration, once the manipulator has reached the near-goal configuration.

We restricted the learning algorithm to change the rule base, while keeping the fuzzification stage of the control system constant. Clearly, however, that stage also represents domain knowledge: how far is "far"? Even though prior domain knowledge allows us to answer this question fairly well, the details could depend on the environment and the requirements under which the system operates. An interesting area of future research would be to investigate a learning algorithm that adapts the fuzzification. Here as before, the neural networks literature provides methods that could be successful in this hybrid system. We consider, for instance, using adapting radial basis functions for the fuzzification stage. Adapting the center and the width of RBFs is a method that has, for instance, successfully been applied to many classification problems [7].

Concluding, our strategy, based on behavioral rules, has led to a transparent system for the navigation of planar robotic manipulators. The rule base can be tuned by hand or implemented as a rule network that is trained by a set of learning commands. The

system is flexible and leads to good performance. We have shown how it can be applied to two and three link manipulators. Importantly, the transparent nature of the navigation system leaves it open for future refinements and additions.

Acknowledgements

The authors thank the referees for their useful suggestions on the content and the presentation of the paper.

References

[1] K. Althoefer, Neuro-fuzzy motion planning for robotic manipulators, Ph.D. Thesis, Department of Electrical and Electronic Engineering, King's College London, 1997.
[2] K. Althoefer, L.D. Seneviratne, P. Zavlangas, B. Krekelberg, Fuzzy navigation for robotic manipulators, Int. J. Uncertainty, Fuzziness Knowledge-based Systems 6 (2) (1998) 179–188.
[3] A. Barto, Reinforcement learning for control, Curr. Opinion Neurobiol. 4 (1994) 888–893.
[4] R.A. Brooks, A robust layered control system for a mobile robot, IEEE J. Robotics Automat. RA-2 (1) (1986) 14–23.
[5] P. Dayan, G. Hinton, Feudal reinforcement learning, Adv. Neural Inform. Process. Sys. 5 (1993) 271–278.
[6] F. Hoffmann, O. Malki, G. Pfister, evolutionary algorithms for learning of mobile robot controllers, Proceedings of the EUFIT '96 — The Fourth European Congress on Intelligent Techniques and Soft Computing, Vol. 2, Aachen, September 1996, pp. 1105–1109.
[7] D. Husmeier, S.J. Roberts, Regularisation of RBF-networks with the Bayesian evidence scheme, Proceedings of the International Conference on Artificial Neural Networks (ICANN99), IEE Press, Edinburgh, September 1999, pp. 533–538.
[8] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, The Int. J. Robotics Res. 5 (1) (1986) 90–98.
[9] B. Kosko, Neural Networks and Fuzzy Systems, Prentice-Hall International, Englewood Cliffs, NJ, 1992.
[10] R. Kruse, J. Gebhard, F. Klawonn, Foundations of Fuzzy Systems, Wiley, New York, 1994.
[11] J.-C. Latombe, Robot Motion Planning, Kluwer Academic Publishers, Boston, 1991.
[12] E.H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, in: E.H. Mamdani, B.R. Gaines (Eds.), Fuzzy Reasoning and its Applications, Academic Press, New York, 1981, pp. 311–323.
[13] P. Reignier, Fuzzy logic techniques for mobile robot obstacle avoidance, Proceedings of the International Workshop on Intelligent Robotic Systems, Zakopane, July 1993, pp. 187–197.
[14] W. Risse, B. Fink, M. Hiller, Multisensor-based control of a redundant SCARA robot, Proceedings of the Ninth World Congress on the Theory of Machines and Mechanisms, Vol. 3, Milan, August/September, 1995, pp. 2037–2041.
[15] H. Roth, K. Schilling, B. Theobald, Fuzzy control strategies for mobile robots, Proceedings of the Ninth Fachgespräch über Autonome Systeme, Munich, October 1993, pp. 251–260.
[16] K. Song, J. Tai, Fuzzy navigation of a mobile robot, Proceedings of the 1992 IEEE/RJS International Conference on Intelligent Robots and Systems, Raleigh, July 1992, pp. 621–627.
[17] M. Sugeno, An introductory survey of fuzzy logic, Inform. Sci. 36 (1985) 59–83.
[18] M. Sugeno, K. Murakami, Fuzzy parking control of model car, Proceedings of the 23rd IEEE Conference on Decision and Control, Las Vegas, December 1984, pp. 902–903.
[19] R.S. Sutton, Planning by incremental dynamic programming, Proceedings of the Ninth Conference on Machine Learning, Morgan Kaufmann, Los Altos, CA, 1991, pp. 353–357.

[20] N. Tschichold-Gürman, The neural network model rule net and its application to mobile robot navigation, Fuzzy Sets and Systems (methods for data analysis special issue) 85 (2) (1997) 287–303.
[21] W.A. Wolovich, Robotics: Basic Analysis and Design, Holt, Rinehart & Winston, New York, 1987.
[22] L. Zadeh, Fuzzy sets, Inform. Control 8 (1965) 338–353.

Kaspar Althoefer holds a degree in Electronic Engineering from the University of Technology Aachen, Germany. He carried out his doctoral research in the Department of Electrical and Electronic Engineering at King's College London where he completed his Ph.D. in 1997. During that time, he worked on the development of neural networks and fuzzy systems as well as of appropriate training algorithms to solve the problems of navigation, obstacle avoidance and path planning for robotic manipulators and mobile robots. In 1996, he was appointed Lecturer in the Department of Mechanical Engineering at King's College London and is since engaged in research on the subject of mechatronics, fuzzy logic and neural networks.

Bart Krekelberg has a degree in Astrophysics (Utrecht University, 1991) and Cognitive Artificial Intelligence (1993). He started work in computational neuroscience with his Ph.D. on nitric oxide and volume learning at King's College London (1997). At the Ruhr-University Bochum he currently works on the experimental side of neuroscience. His main interests are the neuronal mechanisms underlying the visual perception of space and motion. This includes the study of area MST in the macaque monkey. MST is a cortical area involved in the perception of heading and may be instrumental in navigation and obstacle avoidance.

Dirk Husmeier graduated in Physics (Dipl.-Phys.) at the University of Bochum (Germany) in 1991, and recieved both his M.Sc. and Ph.D. degrees in Neural Computation from King's College London in 1994 and 1997, respectively. After working as a postdoctoral research fellow in the Electrical Engineering Department of Imperial College London from 1997 till 1999, he joined Biomathematics and Statistics Scotland (BioSS) as a research scientist in October 1999. His main interests are statistical pattern recognition, supervised learning in neural networks, Bayesian approaches to machine learning, statistical methods for phylogeny, and bioinformatics.

Lakmal D. Seneviratne is a reader in Mechatronics in the Department of Mechanical Engineering at King's College London. His research interests include robotics, automation control, monitoring, and inspection of industrial systems. He is the Director of the Centre for Mechatronics and Manufacturing Systems Engineering at King's College London and the research leader of the Department of Mechanical Engineering.