Experiments in Competence Acquisition for Autonomous Mobile Robots

Ulrich Nehmzow

Ph.D.

University of Edinburgh

1992

Abstract

This thesis addresses the problem of intelligent control of autonomous mobile robots, particularly under circumstances unforeseen by the designer. As the range of applications for autonomous robots widens and increasingly includes operation in unknown environments (exploration) and tasks which are not clearly specifiable *a priori* (maintenance work), this question is becoming more and more important.

It is argued that in order to achieve such flexibility in unforeseen situations it is necessary to equip a mobile robot with the ability to autonomously acquire the necessary task achieving competences, through interaction with the world.

Using mobile robots equipped with self-organising, behaviour-based controllers, experiments in the autonomous acquisition of motor competences and navigational skills were conducted to investigate the viability of this approach.

A controller architecture is presented that allows extremely fast acquisition of motor competences such as obstacle avoidance, wall and corridor following and dead end escape: these skills are obtained in less than five learning steps, performed in under one minute of real time. This is considerably faster than previous approaches. Because the effective wiring between sensors and actuators is determined autonomously by the robot, sensors and actuators may initially be wired up arbitrarily, which reduces the risk of human error during the setting up phase of the robot. For the first time it was demonstrated that robots also become able to autonomously recover from unforeseen situations such as changes in the robot's morphology, the environment or the task. Rule-based approaches to error recovery obviously cannot offer recovery from unforeseen errors, as error situations covered by such approaches have to be identified beforehand.

A robust and fast mapbuilding architecture is presented that enables mobile robots to autonomously construct internal representations of their environment, using self-organising feature maps. After a short training time the robots are able to use the self-organising feature map successfully for location recognition.

For the first time the staged acquisition of multiple competences in mobile robots is presented. First obtaining fundamental motor competences such as wall following and dead end escape (primary skills), the robots use these in a second stage to learn higher levels of competence such as the navigational task of location recognition (secondary skills). Besides laying the foundation of autonomous, staged acquisition of high level competences, this approach has the interesting property of securely grounding secondary skills in the robot's own experience, as these secondary skills are defined in terms of the primary ones.

Kurzfassung

Diese Dissertation befaßt sich mit dem Problem der intelligenten Steuerung autonomer mobiler Roboter, insbesondere in vom Ingenieur nicht vorhergesehenen Situationen. Diese Frage gewinnt zunehmend an Bedeutung, da das Einsatzspektrum autonomer Roboter sich ständig erweitert und in zunehmendem Maße den Einsatz in unbekannter Umgebung (Exploration) sowie *a priori* schlecht parametrierbare Aufgaben (Wartungsarbeiten) umfaßt.

Um Flexibilität in solchen unvorhergesehenen Situationen zu erreichen, ist es notwendig, Robotersteuerungen zu entwickeln, die es dem Roboter erlauben, die benötigten Fähigkeiten autonom, in Interaktion mit der Umwelt zu erwerben — so die hier vertretene These.

Um die Brauchbarkeit dieses Ansatzes zu untersuchen, wurden Experimente mit autonomen mobilen Robotern durchgeführt, die die An-

wendbarkeit von selbstorganisierenden, verhaltensbasierten Steuerungen für den Erwerb von motorischen Fähigkeiten und für Navigation untersucht haben.

Eine Steuerung wird vorgestellt, die das extrem schnelle Lernen von motorischen Fähigkeiten wie Hindernisausweichen, Wandfolgen, Korridorfolgen und Entweichen aus Sackgassen erlaubt. Diese Fähigkeiten werden in weniger als fünf Lernschritten in unter einer Minute Echtzeit erworben. Dies ist erheblich schneller als bisherige Ansätze. Da die effektive Verbindung zwischen Sensoren und Aktuatoren vom Roboter selbst bestimmt wird, können Sensoren und Aktuatoren anfangs in beliebiger Weise angeschlossen werden. Dies vermindert die Anfälligkeit für Bedienfehler während der Installationsphase des Roboters. Erstmalig wurde gezeigt, daß Roboter auf unvorhergesehenen Situationen wie Änderungen der Robotermorphologie, Änderungen in der Umwelt oder Änderungen der Aufgabe reagieren können, ohne ihre Einsatzfähigkeit zu verlieren. Regelbasierte Ansätze zur Fehlerkorrektur sind dazu nicht geeignet, da für solche Ansätze die Fehlersituationen schon von vornherein bekannt sein müssen.

Ein robustes und schnelles Kartographiesystem auf der Grundlage von selbstorganisierenden Merkmalskarten wird vorgestellt, welches es mobilen Robotern ermöglicht, in autonomem Betrieb eine Repräsentation der Umgebung zu erstellen. Nach kurzer Trainingszeit sind die Roboter in der Lage, mit Hilfe dieser Karten eine Ortserkennung erfolgreich durchzuführen.

Erstmalig wird auch der stufenweise Erwerb mehrerer Fähigkeiten durch Roboter präsentiert. Nachdem grundlegende Fähigkeiten wie Hindernisausweichen, Wandfolgen oder Entweichen aus Sackgassen erlernt sind (Primärfähigkeiten), verwenden die Roboter diese in einer zweiten Stufe des Lernprozesses zum Erwerb komplexerer Fähigkeiten wie der der Ortserkennung (Sekundärfähigkeiten). Zum einen ist dies der Grundstein für den autonomen, stufenweisen Erwerb komplexerer Fähigkeiten, zum anderen hat dieser Ansatz die interessante Eigenschaft, daß die Sekundärfähigkeiten des Roboters in den Erfahrungen des Roboters und nicht in von außen vorgegebenen Definitionen verankert sind, da die Sekundärfähigkeiten durch die Primärfähigkeiten definiert sind.

Résumé

Tandis que les applications des robots autonomes mobiles sont de plus en plus nombreuses - par exemple lorsqu'il s'agit déxplorer des territoires largement inconnus (par exemple, Mars) ou de travailler dans des environnements incertains (par exemple pour lutter contre un incendie ou pour travailler dans une centrale nucléaire) - les systémes de contrôle de ces robots vont devoir de plus en plus s'accomoder de circonstances non prévues par leur concepteurs.

Cette thèse traite de ce problème, cést-à-dire de la question de savoir comment un contrôleur de robot mobile peut s'avérer flexible en circonstances imprévues. Alors que les tentatives antérieures pour assurer une plus grande flexibilité en matière de contrôle robotique se sont focaliseés, soit sur des solutions mettant en œuvre des stratégies prédéfinies — destineés à résoudre des situations dérreur connues a priori — soit sur la construction de contrôleurs à partir de modules robustes, indépendants et auto-suffisants, la thèse soutenue ici est que, pour s'accomoder de situations reéllement imprévues par son concepteur, un robot doit pouvoir acquérir des compétences et s'adapter à des circonstances changeantes de facon autonome.

Au moyen des robots mobiles Alder et Cairngorm, qui ont été construits dans le cadre du projet "Really Useful Robots" du

Département d'Intelligence Artificielle de l'Université dÉdimbourg, diverses expériences ont été réaliseés afin dé'tudier les mécanismes qui permettent à des robots mobiles d'apprendre à modifier leur comportement, par essais et erreurs, si cela s'avère nécessaire à la réalisation de leur objectif et à s'auto-adapter en cas de circonstances imprévues. Ce résultat a été acquis grâce à la coopération de composants figés et de composants modifiables au sein des contrôleurs de ces robots, les composants figés mettant en œuvre ce qu'on pourrait appeler des règles instinctives (éxécutant des comportements immuables et prédéfinis) et les composants modifiables étant implémentés au moyen d'architectures de calcul connectionnistes (Réseaux de Neurones Artificiels).

Au cours déxpériences ayant trait à l'acquisition d'aptitudes motrices fondamentales, Alder et Cairngorm ont appris par essais et erreurs à éviter des obstacles, à sé'chapper de culs-de-sacs, à suivre des murs et des corridors. Ces robots se sont avérés capables de s'accomoder de circonstances imprévues telles que des changements dans leur environnement ou dans leur propre morphologie: ils se sont remis de ces changements et ont repris la tâche qu'ils cherchaient à accomplir. D'autres expériences ont également montré que ces robots peuvent exploiter une information contextuelle et agir intelligemment dans des circonstances differentes.

Au cours déxpériences portant sur la reconnaissance de lieux, Alder et Cairngorm ont utilisé des cartes topographiques auto-organiseés comme représentations distribueés de leur environnement. Générés à partir des stimuli déntreé provenant de leurs capteurs sensoriels ou des actions motrices effectueés par les robots, les schémas déxcitation émergents au sein de ces cartes topographiques se sont avérés fidélement corrélés aux emplacements du monde reél. De ce fait, les robots étaient capables de reconnaître les lieux qu'ils avaient déja visités dans leur environnement.

Enfin, au cours de recherches portant sur l'apprentissage par étapes, des expériences sur l'acquisition de compétences motrices et sur la reconnaissance de lieux ont été combineés: Cairngorm a ainsi appris d'abord à éviter les obstacles, à sé'chapper des cul-de-sacs et à suivre les murs puis, après que ces compétences aient été acquises, à les utiliser pour entraîner un systéme à base de carte topographique auto-organiseé à reconnaître les lieux parcourus.

Grâce aux contrôleurs auto-organisés décrits dans cette thèse, il a été montré que des robots mobiles peuvent acquérir de manière autonome les compétences permettant de mener une tâche à bien et qu'ils peuvent s'accomoder avec succès de situations imprévues. Pour la première fois, il a été également montré que des robots peuvent tirer bénéfice de compétences acquises antérieurement pour réaliser ensuite d'autres apprentissages. Un tel apprentissage par étapes est nécessaire pour le développement autonome d'aptitudes plus élaboreés chez les robots.

The composition of this thesis and the research reported in it are entirely my own work, except where otherwise stated.

Ulrich Nehmzow

Acknowledgements

Robotics is fun. I would not have realised this without the exciting and stimulating company of my friends. I am grateful for inspiring discussions I was priviliged to have, most of all with Tim Smithers (I still remember us using the mirror in our hotel room in Kaiserslautern as a whiteboard), and also with John Hallam. I owe much to both: Tim inspired and directed my work and kept me excited, John helped me to settle, guided my first steps and carefully and helpfully analysed and assessed my work.

I am grateful for the many opportunities I was given to meet other people and see their work. I gladly remember the "RUR tour" through Germany: for two weeks Tim, John, Pete and I visited many research institutions there — animated discussions wherever we were shaped our work and gave directions for the future.

My best and most faithful critic from the public was my dear wife Claudia, who supported me in many special ways.

I would not have started and finished this work without the help of the Nehmzow Clan. My mother encouraged me to start it and my father encouraged me to keep going. Thank You to Ama and Max, too.

I thank Martin Waite for animated discussions about robotics, in bothies and elsewhere, KP Naidu, Alistair Conkie and Andrew Fitzgibbon for their help in computer matters all along the way; and Phyllis Garden Richardson and Jean-Arcady Meyer for translating the abstract of this thesis into French.

The simulator of Alder was kindly given to me by members of the "FORANN" project at Bremen University.

Thank You to Peter Ross, Brendan McGonigle, Manfred Knick, Chris Malcolm, Peter Forster and Barbara Webb for helpful comments. Thank You also to the technical workshops, in particular to Tom Alexander, Douglas Howie and Hugh Cameron, and to Gillian Morrice and Karen Konstroffer, who all supported this work much through their practical help.

I gratefully acknowledge the support from both the Science and Engineering Research Council, who supported the work reported here for two years, and from the Department of Artificial Intelligence, which provided a very stimulating and supportive environment.

I dedicate this thesis to

my wife Claudia, my mother and my father, my family.

Contents

Chapter 1

Experiments in Competence Acquisition for Autonomous Mobile Robots

1.1 Introduction

One of the most striking features of intelligent beings is that they can cope with situations they have never encountered before, and that they can apply previously acquired knowledge to such unforeseen situations.

This thesis addresses the problem of coping with unforeseen situations in the inanimate world, in mobile robotics. Are there mechanisms we can successfully employ to achieve flexibility of mobile robots in unforeseen situations, and what are the appropriate means to investigate this research topic? These are the questions I try to answer in this thesis.

When talking about unforeseen situations, I use the word unforeseen in the strict sense of the word. It describes the fact that a situation has not been anticipated by the designer of a robot controller, and it describes a situation that has neither been identified nor been experienced before the actual event.

To be able to achieve this flexibility in unforeseen situations three properties are necessary. The agent must be able to:

1. acquire knowledge in the first place,
2. detect changing circumstances, and
3. find appropriate actions in the unforeseen situation, possibly using previously acquired knowledge.

Why would it be desirable to build robot controllers based on these principles? If such flexibility could be achieved in a mobile robot, it would enhance the robot's capability of staying operational, because the robot could recover even if single components fail. It would increase the robot's versatility, as such a robot could be used even if the world it is going to be used in is unknown to the designer beforehand. Examples of possible applications are the exploration of unknown territory (e.g. planets), performing maintenance work in inaccessible areas (e.g. blocked pipes) or work in dangerous or contaminated environments (e.g. nuclear power stations). Finally, setting up such robots would be easier and less prone to error, as the effective wiring between sensors and actuators could be established autonomously by the the robot itself. The first two points, however, lie in the future — the work reported here starts at the very beginning, at the acquisition of basic competences in mobile robotics. The last point has already been achieved to some extent: for the robots described in this thesis sensors and motors may initially be connected arbitrarily, an effective wiring between them is established by the robots themselves. The questions are these: what are the components required for a robot controller that allow the robot to react flexibly and effectively in an unforeseen situation? Which are the factors in the robot and its controller that make the robot either succeed or fail? Identifying these factors is crucial for building robots exhibiting task-achieving behaviours, even under unforeseen circumstances.

1.2 Tools for Investigation

Before any attempt to answer this can be made, a fundamental point has to be discussed: which are the appropriate means to investigate the subject? The validity of the investigation method used has to be established first, as it is obvious that the investigation method will influence the results. In 1981 Tversky and Kahneman conducted the following experiment on the *framing effect*[1] (after [Matlin 89]).

> Problem 1: Imagine that the United States is preparing for the outbreak of an unusual Asian disease, which is expected to kill 600 people. Two alternative programs to combat the disease have been proposed. Assume that the exact scientific estimate of the consequences of the programs are as follows:
>
> - If program A is adopted, 200 people will be saved.
> - If program B is adopted, there is a one-third probability that 600 people will be saved, and two-thirds probability that no people will be saved.
>
> Out of 152 students given these two options, 72% chose program A and 28% chose program B.
>
> Problem 2: Now imagine the same situation, with these two alternatives:

[1] The effect that the way a question is asked (framed) and the background context of the choice itself can influence the outcome of the decision.

- If program C is adopted, 400 people will die.
- If program D is adopted, there is a one-third probability that nobody will die, and two-thirds probability that 600 people will die.

> Out of 155 different students given these two options, 22% favoured program C, but 78% favoured program D!

The point is obvious: the value of answers received or the validity of results obtained has to be interpreted with respect to the question asked or the investigation method used, respectively.

For scientific research in general, and for research in intelligent mobile robotics in particular, two approaches can be taken: simulating or experimenting[2]. Simulation tries to reproduce what are judged to be just the significant events or processes under test conditions. Initially specifying rules, relationships, operating procedures and initial state of the simulation, the designer of the simulation then tries to relate the results obtained by simulation to the actual process simulated. In the area of autonomous mobile robots simulation almost always means computer-based simulation. Experiment, on the other hand, stands for the actual experiments, the "real McCoy", without prior judgement as to what is significant. In mobile robotics this means the use of mobile robots, operating in a niche of the real world (I say niche, because whatever the world the robot is operating in is like, it is never going to be the whole of the whole world).

To date, in mobile robotics research computer-based simulation has been used far more often than actual experimentation; for what reasons? There are advantages to simulation. It is — or at least was

[2]Throughout this thesis I use the term "experiment" for actual physical experiments, conducted with robots.

until recently — easier to set up. The equipment for the simulation of intelligent agents can be found in every Artificial Intelligence or Computer Science laboratory: the computer. It is often faster than an experiment. It works within clearly defined boundaries, all parameters that influence the simulation are defined by the designer. Simulation of identical environment states is possible, because these states are clearly defined and can be set again and again. This is in fact a major difference between simulation and experiment: the simulation assumes that sufficient information about the current environment state is available, so that the state of the controller can be assumed to be identical to the current environment state. The experiment, on the other hand, shows that identical world states can never be reached again. Only identical controller states are possible — equivalence of world state and controller state cannot be assumed and is not assumed. Because of this fundamental difference, I use different terms throughout this thesis when I talk about simulation and experiment: *agent* and *environment* for simulations, *robot* and *world* for experiments.

As simulation is often easier to set up and cheaper to implement, as it is often faster than experiments and offers very clearly defined boundaries which are easier to analyse, as it offers the possibility to investigate identical environment states again and again in order to find a suitable set of parameters, why have I chosen to conduct *experiments* to investigate the question of competence acquisition and flexibility in unforeseen situations in mobile robotics? The reason is this: simulation is only as good as the data it is based on. In order to build a scientifically useful simulator, one must either

- have sufficient data from previous experiments, or
- use data obtained directly from the world.

Following page: simulated turbine wheel, as used in finite element analysis.

Here is an example for a scientifically useful simulation, based on a wealth of experimentally acquired data: at NEI Parsons generator and turbine manufacturers in Newcastle, simulation of turbine blades under the influence of pressure, high temperature and centrifugal forces has, in some cases, replaced the 'test wheel' which was especially built to be tested under realistic conditions. Whereas building a test wheel took months, a simulation is done in days (a simulated turbine blade for the finite element analysis is shown on page 6). A test wheel cost 250,000 pounds, the computer and software for the simulation a fraction of the price. The simulation used was carefully developed by Rolls Royce, another turbine manufacturer, and yet it was only possible for Parsons to successfully use this simulator because test wheels were available that could be used to fine-tune the simulation. Without the knowledge based on experiments, both at Rolls Royce and Parsons, the simulation of turbine blades would have been of little use for the engineering of new turbines.

Unfortunately, in intelligent mobile robotics this wealth of data from experiments is not yet available, nor is the theory that is also a necessary prerequisite to the building of such simulators. Not all essential parameters that make a robot succeed in its task in the world have been identified yet. This is a difficult problem. It means that it is impossible to "simply concentrate on the most important points" when implementing a simulation, because to date it is not at all clear to us what the most important points are! Obviously, important factors influencing a robot's behaviour will be its dimensions, the signals coming from its sensors and the effects of its actuators. But what about the influences of wear, backlash, fatigue, slippage, friction, stiction, asymmetries and climate? Can these be ignored in a simulation without affecting its usefulness?

Experiments suggest that many of these "minor" aspects of robotics do play an important role[3].

To make things even worse, even including all known aspects will not guarantee that simulated agent and robot will behave identically. The interaction of robot and world produces effects that cannot be ascribed to properties of robot or world alone (figure 1.1 gives a graphical demonstration of this). For example, pushing hard against an obstacle will displace either the obstacle or the robot, thus changing their relative positions.

Often, stochastic processes help the robot to overcome problems, although one would not have expected this from a theoretical analysis. The following result from an experiment highlights this fact: suppose a mobile robot is built that uses two tactile sensors mounted at the front of the robot to steer away from obstacles. Such a robot is shown in figure 1.2. Whiskers reverse the turning direction of a motor when they are on (this is indicated by a "-" in the diagram), which means that the robot will turn left when the right whisker is on, right when the left whisker is on and reverse when both whiskers are on. Such a vehicle is similar to the ones discussed by Braitenberg ([Braitenberg 84]). In Braitenberg's vehicles, sensor excitation is proportional to the action of the motor connected to the sensor, whereas in the example shown in figure 1.2 the sensors are binary sensors, and simply reverse the turning direction of the motors when they are on.

I have programmed "Alder" (a robot) to behave that way and, as expected, the robot turns away from solitary obstacles. Something surprising happens, though, when the robot is put in a dead end, as shown in figure 1.3: instead of turning left and right forever, possibly

[3][Malcolm 91] reports that a particular assembly robot "reliably" failed on Monday mornings, because of slightly smaller dimensions due to being cold!

Figure 1.1: Arcimboldo: The interaction of parts produces effects that are not present in the mere collection of them.

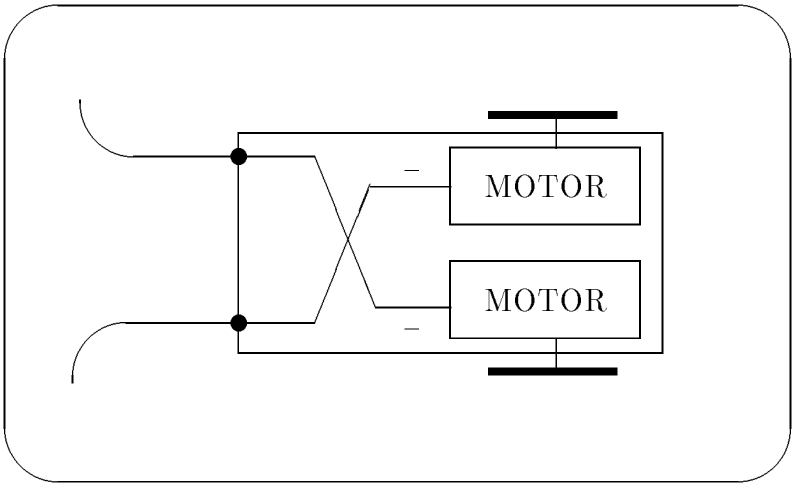

Figure 1.2: Simple, obstacle avoiding vehicle.

reversing occasionally, the robot often (not always) eventually touches the left wall with its right whisker, or the right wall with its left whisker, which will make it leave the dead end.

This is not a result of the wiring between sensors and actuators (in fact it happens despite the wiring, rather than because of the wiring), but is due to effects that stem from the interaction of the particular robot with its world. Alder, for example, does not turn symmetrically to the left and to the right, but has a bias to one side (every robot has such asymmetries). If the dead end is not too tight, this will make the robot turn more towards one direction than turning back in the other, so that eventually the robot will be able to leave the dead end.

This experiment shows that an intelligent agent cannot be seen independently from its task and its environment.

- The robot,
- the task the robot is to achieve, and
- the world

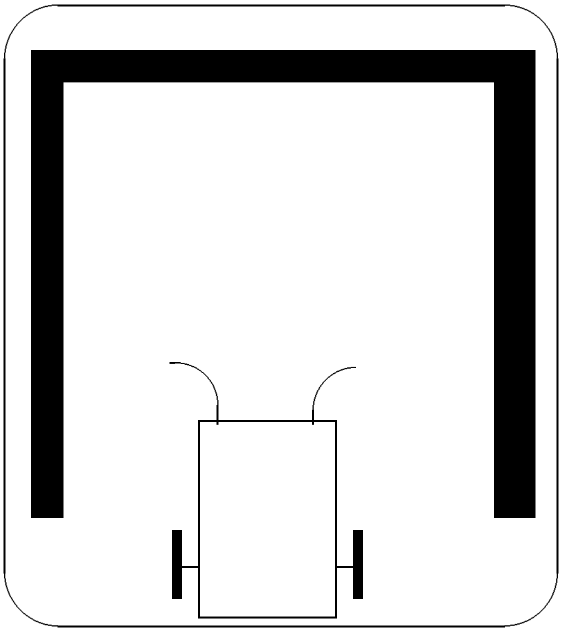

Figure 1.3: Dead end.

have to be seen in perspective. There is no such thing as a general purpose robot, nor indeed a general purpose biological system. Birds, for example, accomplish amazing navigational tasks. The short-tailed shearwater, for instance, migrates across the Pacific from southern Australia as far as the Arctic Ocean, and covers 20,000 to 35,000 kilometers doing so. The shearwater is able to arrive at the desired location by exploiting recurrent wind patterns for navigation ([Waterman 1989]). He is well adapted to his particular task of navigation and the particular environment he is doing it in.

Only if the relevant features in agent, task and world are identified is it possible to build scientifically useful simulations. The conclusion I drew from these considerations was that in order to answer the question of intelligent behaviour of mobile robots, I have to use data obtained from *experiments*, and use this data either immediately in the robot controller, or in some sort of subsequent analysis (which could be a simulator).

1.3 The Robots

For the work reported here I have built two mobile robots, Alder and Cairngorm[4]. A photograph of them is shown after page 18.

Much of the work described in this thesis was done as part of the "Really Useful Robots" project at the Department of Artificial Intelligence at Edinburgh University. The title of the project is apt, although perhaps misleading without explanation. The aim of the project was to investigate mechanisms that give higher degrees of flexibility in mobile robot control, that allow robots to recover from errors without human intervention and to remain operational under changing circumstances, in the presence of noise and variation inherent in the real world. The underlying philosophy of the project was to leave as many decisions as possible to the robot, in other words to avoid predefinition whenever possible, because we believe that predefinition limits the flexibility of robots. The name of the project alludes — somewhat tongue in cheek — to Karel Čapek's play "R.U.R. — Rossum's Universal Robots" (Prague 1920), which introduced the term *robot*[5] to a wider audience, and to "Thomas, the tank engine"[6], where a "really useful engine" is an engine that "gets on with the job, without much fuss".

[4] Ben Alder and Cairngorm are names of Scottish mountains.

[5] From czech *robota*, labour.

[6] A popular British children's book.

1.4 Experiments

Having described the motivation for the work described in this thesis (achieving flexibility in unforeseen situations) and having introduced the tool I used for the investigation (experiments with robots), I can now explain the experiments conducted in a little more detail.

The most important task for mobile robots behaving in some environment is to stay operational whilst interacting with the world they are put in. Without this ability no advanced capabilities can be acquired, no learning or adaptation can take place and no intelligent behaviour can develop. Consequently, the first set of experiments conducted with Alder and Cairngorm concerned the acquisition of basic motor competences that allowed the robots to stay operational in a changing environment.

Using connectionist computing techniques (sometimes also called "artificial neural networks"), the robots, through trial and error, acquire competences such as avoiding obstacles, escaping from cul-de-sacs, following walls and following corridors. They can cope with unforeseen situations such as:

- changes in their morphology, for example swapped whiskers or swapped motor power supplies,
- changes in their task, for example escaping cul-de-sacs after having learned how to avoid single obstacles, and
- changes in their world — for example moving the wall the robot

has been following from one side to the other[7].

A second group of experiments concentrated on the problem of location recognition as a step towards navigation. Again based on connectionist computing techniques, the robots used incoming sensor data to train self-organising feature maps to obtain an internal representation of their *experience space*. Experience space is the appropriate term here, because the representation developed in the self-organising feature map is not a topological representation of the world the robot is operating in, but a representation of the sensor or actuator stimuli (depending on the experiment) the robot receives in the world. The excitation patterns developing in the self-organising feature maps can then be used by the robots to recognise locations within their world. To obtain the sensory input used in the experiments of this second group, a preprogrammed wall following behaviour was used.

The final group of experiments combined the first two. In what I call *staged learning*, the robots first acquire the competence to avoid obstacles and follow walls or corridors, and subsequently use this ability to negotiate mazes or train self-organising feature maps which they can later use for location recognition, as before.

[7]Compare this to the list shown earlier on page 10. Whilst it might seem unlikely that the robot will have whiskers swapped or power supplies reversed in operation, it can be an advantage if a robot equipped with a large number of sensors does not have to be wired up in a particular way. As regards walls swapping sides: In an office environment it is conceivable that the robot encounters situations where the wall on one side of the robot stops, and a wall on the other side starts (door openings on one side), or that the robot changes its direction of travel and thus senses the wall as being on the other side. In warehouses "walls" swap sides all the time if someone else is restacking boxes.

These experiments show that mobile robots can:

1. autonomously acquire task achieving behaviours,
2. detect changing circumstances, and
3. behave intelligently, even under unforeseen circumstances.

The statement I am making through this thesis is thus:

> Coping with unforeseen situations is a relevant problem in mobile robotics, which has received little attention to date. Using connectionist computing techniques, it is possible to achieve reinforcement learning — learning by trial and error through performance feedback — in mobile robots and enable mobile robots to exhibit reliable and robust behaviour, even under circumstances changing in unforeseen ways.

The main conclusion of this thesis is that connectionist computing techniques can indeed successfully be used in mobile robot control, that they allow the robot to acquire competences through trial and error, and that they enable the robot to adjust to changes in the robot itself, its task or its world. This approach increases the flexibility of mobile robots greatly, because it allows

- flexible wiring of sensors and actuators,
- recovery from undesired system states,
- recovery from unforeseen system states, and
- autonomous acquisition of higher level competences in a process of staged learning.

1.5 Organisation of this thesis

Chapter 2 presents work done in mobile robotics to date, relevant to this thesis. It discusses synthetical and analytical approaches to mobile robot control, it describes different ways of using sensor information to control a robot, and it describes mathematical tools that are useful for the task of achieving flexibility in unforeseen situations.

Chapter 3 explains why experimentation is important in robotics and discusses the advantages and problems associated with simulation. Also in this chapter Driesh, a simulation built after Alder, is presented.

Chapters 4, 5 and 6 present the experiments conducted with Alder and Cairngorm, and the results obtained. If flexibility in unforeseen situations is to be achieved, a robot has to be able to assess incoming information independently from predefined knowledge and to autonomously determine useful actions. In chapter 4 I describe experiments concerning the acquisition of basic competences such as obstacle avoidance and wall following. These competences allow the robot to stay operational, even if circumstances change in unforeseen ways. The chapter describes how Alder and Cairngorm cope with changes in their setup (wiring of sensors and actuators) and changes in their environment, and how they regain the ability to act successfully in their environment after an unforeseen event has occurred.

The experiments in competence acquisition described up to this point have one problem: whenever the robots operate under a new context, as for example in moving from open space with single obstacles into a dead end, they have to relearn the appropriate connections between sensor signals and motor signals in order to stay operational. Section 4.7 describes experiments in which Alder and Cairngorm identify the context they are operating in, and by this can operate successfully in more than one context without renewed learning.

Chapter 5 concentrates on a different problem, that of location recognition. Here experiments are performed in which the robots construct an internal representation of their environment using self-organising feature maps. The excitation patterns of the net are used to recognise locations. Three different experiments are presented in total, each of these with a similar controller architecture, but different input stimuli to the self-organising feature map. The first experiment is based on landmark detection, the following two experiments follow an alternative approach and use the motor actions the robot performs in the world instead.

Chapter 6 describes experiments in which the robots first acquire a particular motor skill, and then use this skill to acquire further competences. In a process of staged learning Cairngorm first acquires the skill to avoid obstacles and follow corridors, and then uses this ability to negotiate a maze. In the concluding experiment the competences described in chapters 4 and 5 are combined: having learned obstacle avoidance and wall following, Cairngorm uses these skills to train a self-organising feature map for location recognition.

Finally, chapter 7 recalls the main points of this thesis, raises some questions related to the work presented in this thesis and points to further questions and work still to be done.

The aim of this work is to contribute towards the understanding of how task achieving competences in a mobile robot can be acquired autonomously, rather than being predefined by the designer, and how controllers for autonomous mobile robots can be built that enable the robot to react flexibly and intelligently in a world changing in unforeseen ways.

Following page: Alder (left) and Cairngorm (right).

Chapter 2

Background and Review

2.1 Analytical and Synthetical Approach

The work outlined in section 1.1 draws on research in many areas. Obviously, it is related to previous work done in (mobile) robotics, but machine learning, connectionism, even neuroscience have a part in it, too. To review all work done in all of these areas is impossible — I have concentrated on topics that are directly relevant for the experiments presented in this thesis, on mobile robots, on connectionist computing paradigms I have actually used and on learning algorithms that have been used for similar applications to the ones presented here.

To structure this review, I have divided it according to Alder's and Cairngorm's four major tasks :

1. staying operational,
2. navigation,
3. learning, and
4. coping with abnormal situations.

2.2 Staying Operational

Before any higher level competences such as for example navigation can be achieved in a mobile robot, the robot has to accomplish a much more fundamental task first: to remain functional, to stay operational. For a mobile robot this mainly means to avoid obstacles and to get unwedged if trapped in a dead end. It could also mean to maintain a certain battery charge level, but neither Alder nor Cairngorm are able to autonomously recharge their batteries, nor have I found references in the literature of robots that exploit such an ability.

Of these tasks, obstacle avoidance is the one referred to most often in the literature, because it is such a fundamentally important competence. Collision avoidance is a research topic in its own right, see for example [IEEE 91], where a whole section is dedicated to this topic. Two basic approaches to achieving this goal can be found in the literature: either the robot is controlled by reflex-like behaviours, by a direct connection between its sensors and its actuators, or an intermediate planning process determines which action to take in which situation. The latter is mostly called *path planning*, because the robot's task usually is not only to avoid obstacles, but to move to a goal location. However, often pure obstacle avoidance and path planning are identical, because the goal location for the path planner is simply a point on the robot's path, lying behind an obstacle. These two approaches fall into two paradigms that can be seen in mobile robotics.

On the one hand there is the *analytical approach* ([Nehmzow *et al.* 89]): the overall control task is broken up by the designer into fundamental subtasks, which are then implemented. This is a top down approach. [Brooks 85] calls this the *functional decomposition*: the control task is decomposed into a series of functional units; sensor signals percolate through these units until they generate an ac-

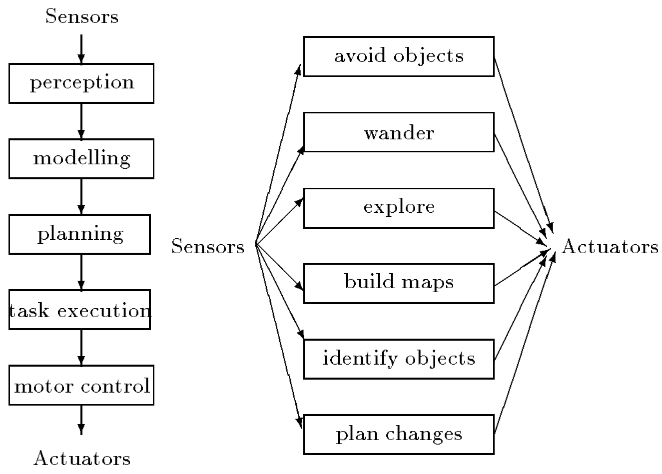

Figure 2.1: Brooks' functional (left) and behavioural (right) decomposition ([Brooks 85], after [Malcolm *et al.* 89]).

tion. Planning the robot's path round an obstacle is an example for an analytical approach. On the other hand, a *synthetical approach* ([Nehmzow *et al.* 89]) combines independent modules to achieve the global control task. This is a bottom up approach. [Minsky 88] and [Steels 88] give examples of synthetical architectures, an example of an implementation in robotics is the behaviour-based *subsumption architecture* ([Brooks 85]): the overall control task is implemented by means of independent, task-achieving behavioural modules working in parallel, and communicating with each other through channels of low bandwidth. Sensor signals enter all modules simultaneously, and all modules may generate robot actions. Avoiding obstacles in a reflex-like manner is an example of such a task-achieving module. Brooks' functional and behavioural decomposition are shown in figure 2.1.

Following a synthetical approach offers advantages: the performance of the controller is not dependent on the weakest link in the chain of functional modules (as in the analytical case); also the synthetical approach lends itself more readily to incremental development of the controller.

The distinction between analytical and synthetical approach is only one aspect of a new, emerging paradigm in robotics. [Malcolm *et al.* 89] characterise this new paradigm by the following set of features:

- Intelligence is emergent through the interaction of the right ingredients, it is not implemented,
- sensing and acting are tightly coupled and not dealt with independently,
- parallelism across the entire range of the robot's information processing functions,
- distaste for symbolic representations,
- active use of the world as its own model,
- minimalism in use of computational power, and
- realisation of the control task by adding independent, self-sufficient components, rather than analysing the whole control task and constructing a monolithic controller (synthetical rather than analytical approach).

2.2.1 Implementations

The *Ipamar* mobile robot ([Kacandes *et al.* 89]) is a good example of the analytical approach. It uses a generalised potential field method and a dynamic path planning algorithm to navigate round obstacles.

Potential field methods ([Khatib 85], [Krogh & Thorpe 86], [Arkin 89]) determine the path between the current position of the robot and the desired position by applying an analogy from physics:

goal locations exert an (imaginary) attractive force, whilst obstacles exert repulsive forces on the robot. By following the gradient of the imaginary potential field the robot moves to the goal location. This method works well when the robot operates in uncluttered space and encounters only convex obstacles; the robot can get trapped, however, in local minima (concave obstacles), also the potential field method can lead to oscillations in the presence of obstacles or narrow passageways ([Koren & Borenstein 91])[1].

The path planning in Ipamar's case works as follows: calculating obstacle positions using ultrasonic sensor readings, a path planning algorithm (BUG) is invoked and determines intermediate goal positions. These intermediate goal positions are then used in the generalised potential field module to determine the new direction of travel vector. The information computed so far can then be used by the SA (set value) module to update the desired vehicle state. All modules communicate with each other via a blackboard.

MACROBE ([Kampmann & Schmidt 89]) is another example for an analytical approach to path planning. The robot is equipped with a predefined map which is updated by the robot's sensor readings. Free motion space and obstacle space are determined by first applying a Delaunay triangulation (which represents the whole environment of the robot in terms of abutting triangles, [Lee & Preparata 84]) to the given map and then converting this Delaunay triangulation into a Voronoi diagram (the straight-line dual to the Delaunay triangulation, [Lee & Drysdale 81]), the latter being a more efficient representation

[1][Barraquand & Latombe 90] overcome these problems by using random motions, [Praßler & Milios 90] by applying a highly parallel, localised algorithm.

than the former[2]. In order to obtain free motion space, the map is then superimposed onto the Voronoi diagram, resulting in a representation of free motion channels and obstacle space which is used by the robot to find its way. This description is very brief, but even from so short a summary it becomes clear that this is a computationally expensive process: for navigation in a pilot plant (whose map has 845 vertices, plus 53 edges and 168 further vertices added by the process) it takes 82 seconds to compute the Voronoi diagram and another 11 seconds to compute the motion channel on a MicroVAX II using MODULA II.

Obstacle avoidance can also be achieved without using symbolic representations of the world in so-called world models : in the early 1950s W. Grey Walter ([Walter 50, Walter 51]) presented his mobile robots *machina speculatrix* and *machina docilis*. These were small robots, not unlike Braitenberg's vehicles ([Braitenberg 84]), that showed interesting and unpredictable behaviour because of the variation inherent in the components they were made of and in the world they interacted with. They were controlled purely by analogue electronic circuits (using valves) and exhibited a reflex-like behaviour. By means of decaying or refreshed charges in a capacitor *machina docilis* was even able to associate stimuli with motor actions. From the work with *machina docilis* Walter deduced a rule that still applies to all learning machines: "Extreme plasticity cannot be gained without some loss of stability". Walter's robots showed many features of the new paradigm in robotics, mentioned above.

Today, the world's fastest road vehicle controlled solely by computer follows the new paradigm, too. *VaMoRs*, a computer vision guided van, is able to travel autonomously on country roads at speeds

[2]The Voronoi tessellation divides a space into polyhedral regions; the boundaries of which are perpendicular lines to the lines joining distinct points of the map ([Hertz *et al.* 91]).

of up to 60 km/h and on motorways at speeds of up to 100 km/h ([Dickmanns & Christians 89]). Stopping in front of obstacles has been successfully demonstrated for speeds up to 40 km/h. No internal world model is used, instead a control oriented approach (applying linear system theory) is pursued, using the world as its own model.

The Subsumption Architecture

The most prominent advocate of robotics following the new paradigm is perhaps Rodney Brooks at MIT. He disagrees with "classical" Artificial Intelligence (AI) approaches, based on the *symbol system hypothesis* ([Simon 69]), which, according to [Brooks 90b], states that intelligence operates on a system of symbols which represent entities in the world. Rather, he supports an approach based on the *physical grounding hypothesis*, which states that in order to build an intelligent system it is necessary to have its representations grounded in the real world. His argument is this ([Brooks 90b]):

> "Without a carefully built physical grounding any symbolic representations will be mismatched to its sensors and actuators. ...Our current strategy is to test the limitations of the physical grounding hypothesis by building robots which are more independent..."

"Internal world models which are complete representations of the external environment", he argues, are not only impossible to obtain, "they are not [even] at all necessary for agents to act in a competent manner" ([Brooks 91b]). Instead of taking the classical AI approach of conducting search through an internal world model — the agent essentially being a problem solver working in a symbolic abstracted domain — he proposes a different approach "where a mobile robot uses the world as its own model, continously referring to its sensors rather than to an internal world model" ([Brooks 91b]).

To use these ideas to build actual robot controllers, Brooks decomposes the whole control task into layers of competence, such as

0) Avoid contact with objects.
1) Wander aimlessly.
2) Explore the world.
3) Build a map.
4) Notice changes in the environment.
etc.

and implements each layer of competence by using an augmented finite state machine (AFSM) dedicated to that particular layer ([Brooks 90b, 3.2.1]). This is a behaviour-based approach to control. Once built and debugged, a layer is, according to Brooks, never altered ([Brooks 85, II.B.]). These AFSMs communicate with each other through channels of low bandwidth (typically 8 or 16 bit words), constantly broadcasting messages and constantly listening to them (but, as Brooks points out, missing up to ten out of eleven messages; the functioning of the controller does not depend upon each message being received and acknowledged). Because higher level behaviours can inhibit (subsume) lower level behaviours, this architecture is called the subsumption architecture ([Brooks 85],[Brooks 91b, sect.6]). The subsumption architecture thus is a distributed control architecture, without internal world model in the classical AI sense (it does have internal representations of states of the finite state machines, though; but it does not have traditional AI representation schemes, nor does it have explicit representations of goals, [Brooks 91b]).

How higher level behaviours can subsume lower level behaviours is explained by an example given in [Brooks 85]. First a controller is built that achieves level 0 behaviour (avoid contacts with objects). This controller uses a number of modules, one of these is the Runaway module that takes its input from the Feelforce module. The Feelforce module considers each detected object as a repulsive force and computes the resulting repulsive force of all detected objects. The Runaway module monitors this 'force' and sends appropriate commands to the Turn module, which controls turns of the robot. If level 1 behaviour (wander) is to be achieved, two modules, Wander and Avoid are added. The Wander module generates a new heading every ten seconds and sends this to the Avoid module. The Avoid module takes this input, as well as the Feelforce input and computes a new direction, which it sends to the Turn module, suppressing the output of the Runaway module. The Runaway module is subsumed.

A good overview of robots with subsumption architecture based controllers built at MIT is given in [Brooks 90b]. The simplest robots have just three thresholded infrared sensors (*Tom and Jerry*), running a three layer subsumption program. *Herbert*, on the other hand, had thirty infrared proximity sensors, a laser light striping system and a magnetic compass ([Connell 89, Brooks 91b, Brooks *et al.* 88]). The robot ran a fifteen layer subsumption program and was able to detect objects having the shape of a drinks can, pick them up with its onboard manipulator and return to the starting position (the navigational system of Herbert is described in section 2.3).

Mapbuilding was demonstrated with *Toto* (see discussion in section 2.3), and learning within a behavioural module with *Genghis*, a legged robot (see discussion in section 2.4).

Discussion of the Subsumption Architecture

The work presented in this thesis has much in common with Brooks' approach. It is behaviour based, it does not use traditional AI representation schemes such as world models, and it does not manipulate symbols which represent entities in the real world in order to determine the robot's actions; rather it uses the world as its own model (as Brooks), referring to sensors, not to symbols.

Both Brooks' approach and the work presented in this thesis produce reliable and robust robot behaviour whilst being computationally cheap. This makes them attractive for building small robots, where computing power is limited because of limited payloads. Differences, however, are found in the actual controller architectures. In Brooks' case, the synthetic step, the composition of an overall behaviour from independent modules, takes place in the designer's mind. The actual layering into independent layers of competences is a manual process, performed during the construction phase of the controller by the designer ([Brooks 85, II.B.]). The controller presented here, unlike Brooks', attempts to allow the autonomous acquisition of task achieving competences: the synthetic step takes place in the robot controller.

This difference is, I believe, an important one. Brooks reports that it is not yet possible to autonomously acquire complete behaviours, using the subsumption architecture ([Brooks 91a, sect.1]):

> "Progress in learning new behaviours has proven difficult. Today, we are constrained to programming each new behaviour by hand."

Alder and Cairngorm, however, *are* able to acquire new behaviours (even if the designer has not thought of them!). The dead end escape behaviour described on page 49 is an example for this [3].

2.3 Navigation

Staying operational, wandering around and avoiding getting stuck are competences that suffice whenever the robot's task is to explore an

[3] Here the robot learns that it is quicker to leave a dead end by always turning in one and the same direction, whenever a whisker is on, rather than performing an obstacle avoidance behaviour, in which it turns away from a signalling whisker.

unknown environment. For a Mars rover, as an example, it is sufficient to transmit pictures to Earth from wherever it is, and to cover as large an area as possible (unless the task is to explore a specific area of Mars). For any task that requires the robot to recognise locations and to reach specified places — delivery tasks are an example — these skills are not enough. Besides staying operational, the robot needs to have the navigational skills of location recognition and goal-directed movement. This implies that the robot needs to have an internal representation of its world (commonly called a map, although it needn't be a map in the sense of a geographical map), and the ability to use this for navigation (if you can't map-read, a map is of no use at all).

Concerning maps, two main approaches can be found in the literature: either a predefined map is used, or a map is built as the robot explores its environment. Sometimes a predefined map is updated, using information gathered during the exploration, too.

[Skewis *et al.* 91] present an example of a robot using a predefined map. *HelpMate* is a mobile robot whose task it is to deliver off-schedule meal trays in Danbury hospital. The layout of the hospital corridors are supplied to the navigational system of HelpMate by the designer. Path planning is unnecessary because the robot's only task is to link two fixed locations in the hospital (the kitchen and the nursing unit), the path to be taken is also predefined by the designer by means of a mathematically defined curve. Two problems remain for the robot: to find out whether it still is on the prescribed path or not, and to circumnavigate any obstacles along the way. To do the former, the robot relies on dead reckoning (using the measurement of left and right wheel angular displacement), corrected by landmark detection (walls and ceiling lights can be detected by the robot's sensory system). For the latter, the robot will first halt for a while if an obstacle is detected. If the obstacle disappears (a person, for example), the path is resumed.

If the obstacle stays, HelpMate uses its sensors to determine whether it can go past the obstacle on the left or the right hand side, or not at all (in which case it will halt until the obstacle is removed). To accomplish a specific task in an *a priori* known environment, a predefined map is the best method to use. It is efficient and usually easy to implement. If, however, unforeseen changes occur, either to the environment (in HelpMate's case this could be the closure of a hallway, for example because of cleaning work), or the task (travelling between two different locations), or the robot (change of sensors, in HelpMates's case for example ceiling light being switched off), then such a scheme will fail completely.

Consequently, work has been done on the autonomous acquisition of maps, rather than using predefined maps. [Knieriemen 91], [Knieriemen & v.Puttkamer 91] present one example of this. In a three-stage process, *MOBOT III* constructs a world model, based on the readings from its 3D laser range camera. First, raw sensor data is analysed statistically and abstract, geometrical features (such as the orientation of the robot in relation to the main features within its world, usually the walls) are extracted from them. This results in a description of the structure of walls, obstacles etc., depending on the momentary position of the robot. In the second stage, views from different positions of the robot or obtained by different sensors are incorporated into one consistent world model. This process is called *sensor fusion*. The resulting map is comparable to a geometrical, predefined map, as was for example used in HelpMate's case. The last stage extracts a symbolic representation from this geometric map (rooms, free motion space between rooms, pieces of furniture etc), which can be used for tasks of a higher level ("go from room abc to room xyz" etc.).

By constructing a map, for example in the way used by [Knieriemen & v.Puttkamer 91, Knieriemen 91], the problems that occur when something changes can be avoided. In such a case the map is simply changed as well. Obviously the ability of the robot to reliably navigate depends on the quality of the map. In the case of a predefined map this is no problem (I assume that the designer of the robot is able to measure the environment accurately). In the case of an autonomously constructed, geometrical map this is more difficult. In order to incorporate new sensor data into the map for example, the exact position of the robot has to be known ("A position determination as precise as possible is an essential goal when designing an AGV" ([Knieriemen 91, p.93]), otherwise the navigational system will not be able to put the new data "in the right place". Mobot and HelpMate use dead reckoning and landmark detection for this. This problem in constructing a geometrical map can be overcome if a topological map is constructed instead. Such a topological map represents topological relationships (for example neighbourhood relationships) between locations in the robot's world, but it does not represent the exact distances and angles between these locations. The demands on sensor accuracy are therefore less if a topological map is to be constructed.

[Mataric 91] presents a navigational system that uses such a topological representation of the world. Using three layers of competence in a subsumption architecture based controller, the mobile robot *Toto* explores its environment by following boundaries on its left hand side. At junctions, random perturbations let the robot explore different paths at different times ([Mataric 92]). Using ultrasonic range finders and a compass, the robot identifies landmarks along its path and labels them according to their type (left wall, corridor and long irregular boundary) and the compass heading of the robot. To construct a distributed spatial representation of its environment, Toto assigns a node on the map to each landmark, linked via an arc to the preceding and succeeding landmark. The robot's current location is identified by an activated network node, a path to a goal loacation is found by spreading activation from the goal node evenly in all directions, until the activation reaches the current robot position (this is similar to the reaction diffusion dynamics proposed by Steels ([Steels 88])).

This scheme is perhaps the closest to the one used on Alder and Cairngorm. Both Mataric's approach and the one presented in chapter 5 (see also [Nehmzow & Smithers 91a]) are robust and computationally inexpensive, generate topological rather than geometrical maps and do not use any centralised reasoning agent. In both cases the current robot location is represented by an activated node or network region, respectively, and both have been implemented on robots governed by behaviour-based controllers.

Apart from the fact that Mataric claims that this scheme is modelled after the mapping function of the rat's hippocampus (a claim not made in this work) the biggest difference is in how features to be mapped are detected: in Toto, mapable features are identified by their type and the compass heading of the robot, whereas in Alder and Cairngorm they are recognised by the history of preceding events (either previous landmarks or previously executed motor actions), i.e., by the arriving at a location.

[Mataric 91] states that each landmark is associated with a motion directive, indicating the required motion to reach the next landmark, thus addressing the question of map interpretation. These motion directives are derived from the compass bearing associated with that landmark. Because each landmark is labelled twice (once for each of the two travelling directions of the robot) Toto can turn round if that is the shorter way to a goal location ([Mataric 92]).

As soon as a landmark with an identical label to a previously encountered landmark is detected, it is assumed that one whole circuit is completed and the landmark is not mapped. This means that the algorithm is crucially dependent on the correct labelling of landmarks: variations in sensor readings will result in new labels, and thus in new nodes of the map. Contrary to this, the mapbuilding scheme presented in this thesis is markedly more robust in the presence of noise and variation because it uses the topological mapping occurring on the self-organising feature map (SOFM): input vectors generated at identical physical locations, but differing slightly because of sensor variation, will excite neighbouring areas of the SOFM, resulting only in a slightly different response of the network. This has been verified experimentally, too. Although input vectors often vary at identical locations, locations are nevertheless identified correctly.

That simple navigation is possible without any map-like structure at all was shown in [Brooks 91b, 6.2]. Herbert, the can collecting robot, finds its way back to the starting location by using rules like "when passing through a door southbound, turn left". Different sets of rules were applied depending on whether the robot's gripper was closed (can in hand) or open (no can in hand), so that the robot would retrace its original path as soon as the gripper was closed.

2.4 Learning

Being independent of predefined knowledge increases a robot's flexibility not only when it comes to mapbuilding (as discussed in the previous section), but also when other tasks have to be accomplished. For example, if the robot is able to autonomously determine the best way to interpret sensor signals in order to stay clear from obstacles, it can cope with an *arbitrary* initial sensor connection: one potential source of error less! Similarly, if things change (for instance, if in the case of a wall following robot the wall stops on one side and continues on the other) a robot that is able to learn could adapt to the new situation.

Sometimes the best (or even *any*) way to solve a particular problem is unknown to the designer. In those cases it is hard (impossible) to equip the robot with the necessary *a priori* knowledge to accomplish the task. A learning robot is a way out in such situations.

Whereas in the areas of staying operational and navigation many examples from robotics can be found, this is not the case for learning. Very few robots have been built that show a learning capability.

Shakey ([Nilsson 69]) is an early example of a robot that could, to a certain extent, learn: it was able to store sequences of actions as a whole in so-called *triangle tables*, for later use. These master plans (called MACROPS, for MACro OPerator) effectively became new operators which were used by PLANEX, a planner that used the whole or part of a MACROP to solve a particular problem. Although being able to generalise and use once learned plans subsequently, Shakey was unable to cope with situations that could not be expressed in terms of its predefined operators. (More details about the use of these triangle tables are given in [Fikes *et al.* 72]).

In the following, I will discuss both actual implementations of learning algorithms in robotics as well as algorithms that might be used for that purpose, but have not yet been tried on robots.

2.4.1 Reinforcement Learning

Reinforcement learning techniques are particularly suitable for robotic applications in which mistakes of the robot are not immediately fatal and where some sort of evaluation function of the robot's performance exists. Reinforcement learning uses such an overall performance measure (the reinforcement) to control the learning process ([Barto 90, Torras 91]), in this it differs from supervised learning schemes (for example certain kinds of connectionist computing architectures), which use specific target values for individual units. This property can be particularly useful for robotics, where often only the overall desired behaviour of the robot is known; however at the same time this can also be a problem, as it can be difficult to establish which parameter within the controller to alter in order to increase the reinforcement. [Sutton 91]:

> [The term reinforcement learning covers techniques of] "learning by trial and error through performance feedback, i.e. from feedback that evaluates the behaviour, ... but does not indicate correct behaviour."

It is through this performance feedback that a mapping from state (the representation of a particular situation) to action is learned.

[Sutton 91] gives an overview of reinforcement learning architectures for intelligent agents. The simplest of the architectures is the *policy only* architecture, in which the policy of the agent is the only modifiable structure. These architectures work well only if the rewards are distributed around a baseline of zero (that is positive reinforcement is a positive number, negative reinforcement a negative number, they can't both be positive, with the former being bigger than the latter). *Reinforcement comparision techniques* use a prediction of the reward as the baseline and are thus able to cope with rewards distributed around a non-zero baseline. To take non-immediate rewards into account (which neither of these architectures can), the *adaptive heuristic critic* architecture uses a predictor of return (the long term cumulative reward), not reward. In *Q-learning* the predicted return is a function not only of state, but also of the action selected. Finally, *Dyna architectures* are reinforcement learning architectures that contain an internal world model. For each single step of selecting an action and performing it in the real world, Dyna architectures perform another k steps using the world model (k is an integer number).

[Sutton 91] reports that in a simulation on path finding where start and goal location are sixteen squares in a 9x6 grid apart, it takes a Dyna-adaptive heuristic critic 4 steps in the simulated world and another 100 steps (k=100) per each of those four steps to find the path. If a new obstacle is placed in the way, the new path is found in a "very slow" process. He also presents a simulation of a Dyna-Q system that has to find a path of length 10 (squares). This takes 1000 time steps, and another 800 after an obstacle is moved in the way. At k=10 this means 100 steps in the simulated environment, and another 80 to find a new path. For a real robot this can be too slow a learning rate. The only cost to be paid in simulation is that of computing time; in robotics however the cost function is a different one: apart from the fact that due to their battery capacity robots only operate for a certain length of time — the Lego robots used in Edinburgh ([Donnett & Smithers 91]) typically operate for about twenty minutes before their batteries have to be recharged[4] — certain competences such as obstacle avoidance *have* to be acquired very quickly in order to ensure safe operation of the robot.

The conclusion is: for mobile robotics it is crucial that the learning algorithm is fast (on the slow speed of reinforcement learning see also [Brooks 91a]).

The fact that reinforcement learning can be extremely slow is shown by other researchers, too. [Prescott & Mayhew] simulate the AIVRU mobile robot and use a reinforcement learning algorithm similar to the one described by [Watkins 89]. The sensor input space of the simulated agent is a continuous function (this is an unusual feature in reinforcement learning), simulating a sonar that gives distance and angle to the nearest obstacle. It should be noted, however, that a sensor as de-

[4] Alder and Cairngorm last for three to four hours, but because they are very slow they can still only achieve a certain number of "steps".

scribed in [Prescott & Mayhew] does not exist in reality — a sonar does not give readings like the ones used in this simulation! The simulated world is 5m x 5m in area, the simulated robot 30cm x 30cm. Without learning, the agent runs into obstacles in 26.5% of all simulation steps, after 50,000 learning steps (!) this rate drops to 3.25%.

[Kaelbling 90] compares several algorithms and their performances in a simulated robot domain. The agent has to stay away from obstacles (negative reinforcement is applied if it hits an obstacle), it receives positive reinforcement if it moves near a light source. Kaelbling reports that all reinforcement learning algorithms investigated (Q-learning, interval estimation plus Q-learning and adaptive heuristic critic plus interval estimation) suffered from the fact that the agent often acquired an appropriate strategy only very late in the run, because it did not come near the light source in the early stages of the learning process and thus did not receive positive reinforcement. After 10,000 runs, the different algorithms obtained average reinforcement values of 0.16 (Q-learning), 0.18 (interval estimation plus Q-learning) and 0.37 (adaptive heuristic critic plus interval estimation). A handcoded "optimal" controller obtained 0.83. As in the case mentioned earlier ([Prescott & Mayhew]), learning took a long time, and the achieved performance was far below optimal performance.

Slow learning rates, finding the appropriate critic for the reinforcement learning architecture and determining how to alter controller outputs in order to improve performance are the main problems when implementing it on robots ([Barto 90]). Another "problem that has prevented these architectures from being applied to more complex control tasks has been the inability of reinforcement learning algorithms to deal with limited sensory input. That is, these learning algorithms depend on having complete access to the state of the task environment" (([Whitehead & Ballard 90]). For robotics applications this is unrealistic and extremely limiting. Neither of these conditions are assumed for the work reported in this thesis. Perhaps not surprisingly, to date there are only very few implementations of reinforcement learning controllers on real robots. Two examples are described below.

[Mahadevan & Connell 91] present a mobile robot (called Obelix) which uses reinforcement learning (Q-learning) to acquire a box-pushing skill. In order to overcome the credit assignment problem[5], the overall task of box-pushing is divided into three subtasks: box-finding, box-pushing and unwedging. These three tasks are implemented as independent behaviours within a subsumption architecture, box-finding being the lowest level and unwedging being the highest level. Obelix has eight ultrasonic sensors and one infrared sensor, in addition to that the robot can monitor the motor supply current (which gives an indication whether the robot is pushing against a fixed obstacle). Instead of using the raw data, Mahadevan and Connell quantise it into an 18-bit-long vector which is then reduced to 9 bits by combining several bits. This 9-bit input vector is used as an input to the Q-learning algorithm. The possible motor actions of Obelix are restricted to five: forward, left turn, right turn, sharp left turn and sharp right turn. [Mahadevan & Connell 91] present very interesting results that confirm the findings mentioned earlier, as well as observations made in the experiments reported in this thesis. After a training time of 2000 learning steps, the find-box behaviour obtained an average value of reward of 0.16, whereas a handcoded box-finder obtained *circa* 0.25. This confirms that Q-learning requires a large number of learning steps. This is only a confirmation of findings mentioned earlier, but two other aspects are very interesting in relation to the work presented here.

[5]How does one correctly assign credit or blame to an action when its consequences unfold over time and interact with the consequences of other actions ([Barto 90])?

Firstly, [Mahadevan & Connell 91] use a hierarchy of independent behaviours, expressed as behavioural modules within the subsumption architecture. A similar hierarchy is implicitly present in the so-called instinct-rules that govern the process of motor competence acquisition of Alder and Cairngorm[6]. As soon as an instinct-rule[7] is violated, it dominates the robot behaviour until it is satisfied again. It is the nature of these instinct-rules that generates the hierarchy; the "move forward!" instinct-rule for example has the lowest priority, because it is only active as long as no other instinct-rule is violated.

Secondly, the input and output spaces of Obelix are kept small, as is the case for the experiments discussed here (the motor action repertoire of the robots I used is restricted to four actions: forward, left, right and backward move; in some experiments it is further restricted to left turn and right turn). The input space to Obelix' learning algorithm is not only kept small, it also contains quantised information (sonar readings are grouped according to their "range bin"). This preprocessing of sensor data is, I believe, essential for making reinforcement learning algorithms in robots work. I have used quantisation as well, for example to use Alder's sonar sensor for obstacle avoidance (see page 105), or in constructing the input vector for the robots' mapbuilding process (see page 156). In general it is dependent on the actual domain how quantisation can successfully be used.

[Maes & Brooks 90] present a six-legged robot — Genghis — which, again by reinforcement learning, learns to coordinate its leg movements so that a walking behaviour is achieved. Unlike [Brooks 86], who determines the arbitration between behaviours by hand, in Genghis the 'relevance' of a particular behaviour is determined through a statisti-

[6]This process is described in chapter 4, beginning on page 77.

[7]An example of a set of instinct-rules (the ones used to generate wall following behaviour) is shown on page 108.

cal learning process. The stronger the correlation between a particular behaviour and positive feedback, the more relevant it is. The more relevant a behaviour is in a particular context, the more likely it is to be invoked. In Genghis' case positive feedback signals are received from a trailing wheel that serves as a forward motion detector, negative feedback is received from two switches mounted on the bottom of the robot (the switches detect when the robot is not lifted from the ground). As all reinforcement learning schemes, this scheme can only work if the search space in which suitable actions are sought is small. [Kaelbling 90] writes that in experiments with the mobile robot *Spanky*, whose task it was to move towards a light source, the robot only learned to do this successfully if it was helped in the beginning, so that some positive feedback was received. Similarly, in Genghis', Alder's and Cairngorm's case the search space is small enough to give the robot positive feedback at an early stage. Also, like in Obelix', Alder's and Cairngorm's case the input space of Genghis' learning algorithm is small.

[Daskalakis 91] has replicated and extended the experiments in motor competence acquisition discussed in this thesis in chapter 4, using Lego robots ([Donnett & Smithers 91]), which are faster than Alder, and have different sensors, too (infrared and tactile sensors). His robot sucessfully learns to avoid obstacles and follow walls, even under changing circumstances.

All these experiments in reinforcement learning[8], including the experiments described in this thesis, as well as some of the simulations ([Kaelbling 90], for example, uses a 5-bit input vector for the light-finding task) show the following two features:

[8][Shewchuk & Viola 90] have built a subsumption-based learning system in a robot which shows similar properties.

- Small input space, possibly achieved through quantising raw sensor data, and
- small output (motor action) space, possibly achieved through limiting the number of motor actions (as in Obelix, Alder and Cairngorm).

2.4.2 Connectionism

Connectionist computing architectures (also called "artificial neural networks") are mathematical algorithms that are able to learn mappings between input and output states through supervised learning, or to cluster incoming information in an unsupervised manner. Their characteristic feature is that many independent processing units work simultaneously, and that the overall behaviour of the network is not caused by any one component of the architecture, but is emergent from the concurrent working of all units. Because of their ability to learn mappings between an input and an output space, to generalise incoming data, to interpret (cluster) input information without supervision (i.e., without teaching signal), their resistance to noise and their robustness (the term *graceful degradation* describes the fact that the performance of such networks is not solely dependent on the individual unit — losing one will merely mean a degradation, not a total loss of performance) connectionist computing architectures can be used well in robotics. [Torras 91] gives an overview of (largely simulation) work that has been done in the field: supervised learning schemes have been applied to the generation of sequences, both supervised and unsupervised learning schemes have been used to learn non-linear mappings such as inverse kinematics, inverse dynamics and sensorimotor integration, and reinforcement learning has largely been used for tasks involving optimisation, such as path planning.

Simple connectionist computing architectures achieve very fast learning and therefore overcome the problem of slow learning found in some reinforcement learning architectures. The *Perceptron* ([Rosenblatt 62]) is an example. Unfortunately, those fast learners cannot learn arbitrary mappings between inputs and outputs, but only functions that are linearly separable ([Minsky & Papert 88]). Since Alder and Cairngorm only have to learn linearly separable functions (see section 4.1.6), I could nevertheless successfully use this paradigm for learning appropriate reactions of the robots to sensor input stimuli.

Besides offering supervised learning of mappings between input and output space, some connectionist computing architectures can learn in an unsupervised manner (the self-organising feature map (SOFM) is an example): the network structures the incoming data in such a way that a topological representation of the input space is achieved. This can be used to detect structure in the input space; Alder and Cairngorm use a self-organising feature map to represent the world they operate in. After an initial training phase the excitation patterns of the network are used to recognise locations the robots have visited before.

In the following paragraphs I will describe the mathematics of the two networks used for the experiments use on Alder and Cairngorm, the Pattern Associator (which is a kind of Perceptron) and the self-organising feature map.

Perceptron and Pattern Associator

The Perceptron ([Rosenblatt 62]) is a two-layer artificial neural network that is easy to implement, low in computational cost and fast in learning. It consists of two layers of computational units: the input and the output layer (see figure 2.2). The fact that it learns very quickly is bought at the price that it can only learn functions that are linearly separable ([Minsky & Papert 88]), as said before. The functions Alder and Cairngorm had to learn in the particular experiments discussed in this thesis are linearly separable (see page 84), it was therefore possible to use the Perceptron (or rather, a Pattern Associator, a network very similar to the Perceptron) in these experiments.

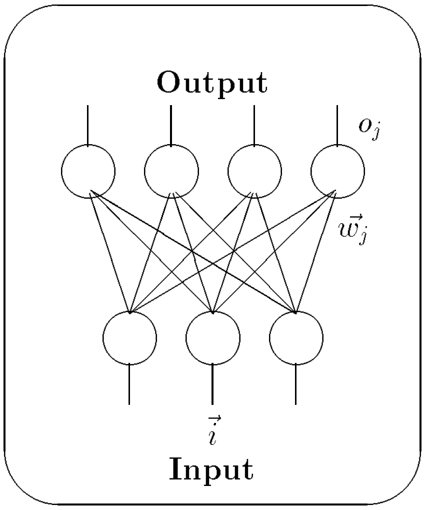

Figure 2.2: Pattern Associator.

The function of input and output units is as follows: input units simply pass the received input signals $\vec{\imath}$ on to all output nodes, the output o_j of output unit j is determined by

$$o_j = \sum_k w_{jk} i_k = \vec{w_j} \cdot \vec{\imath} \tag{2.1}$$

where $\vec{w_j}$ is the individual weight vector of output unit j. If the desired output, the target vector $\vec{\tau}$, is known, updating the weight vectors is done according to the following rule:

$$\Delta \vec{w_j} = \eta(\tau_j - o_j)\vec{\imath} \tag{2.2}$$

$$\vec{w_j}(t_{k+1}) = \vec{w_j}(t_k) + \Delta \vec{w_j} \tag{2.3}$$

η is the so-called gain or learning rate, a parameter that determines how big the changes to the weights are, and therefore how quickly the net learns. A big η (for example 0.8) will result in a network that adjusts very quickly to changes, but which will also be "neurotic" (it will forget all it has learned and learn something new as soon as a couple of freak signals occur). A small η (for example 0.1), on the other hand, will result in a "lethargic" network that takes a long time before it learns a function.

How the target vector $\vec{\tau}$ was determined in the experiments described in this thesis is explained in chapter 4.

In the Perceptron, the real-number output of each node is thresholded, so that the final output is binary. In the Pattern Associator, this thresholding is not done, the output of each output unit in a Pattern Associator is therefore a real number.

Self-Organising Feature Maps

Unsupervised learning can be achieved, using self-organising networks ([Willshaw & v.d.Malsburg 76],[Kohonen 82a],[Kohonen 88], [Ritter *et al.* 89]). These networks develop internal representations of their input space by mapping distinct input vectors onto distinct areas of the network. Figure 2.3 shows an example of a two-dimensional self-organising network.

Self-organising feature maps have several properties that make them interesting for robotics applications: firstly, they require no explicit teaching signal in order to structure the input signal space. Instead of associating particular input stimuli with particular output states (which have to be defined from outside, hence supervised learning) — as Perceptron, Pattern Associator (see previous section) and Backpropagation network[9] (see appendices 7.3 and A) do — self-

[9] The Backpropagation network allows the supervised learning of arbitrary func-

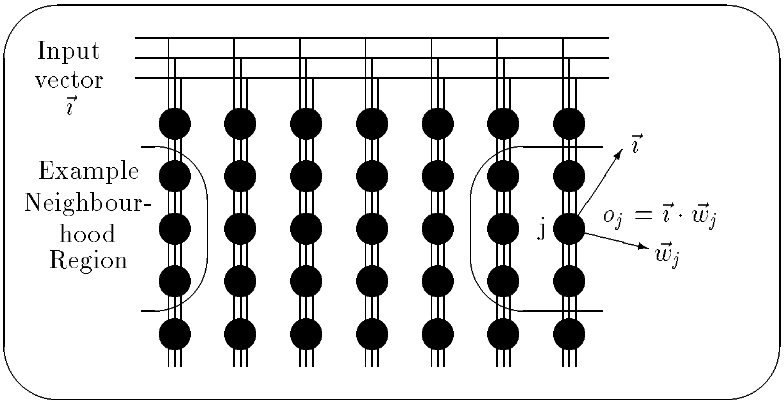

Figure 2.3: A two-dimensional self-organising network.

organising feature maps are able to represent statistical information contained in the input space and cluster incoming data in such a way that topological relationships between input signals are preserved, without external teaching signal. In particular, the resulting map becomes aligned along those directions in the input signal distribution with the most significant variance , and the point density of the weight vectors of the map approximates the probability density function of the input vectors ([Kohonen 82a],[Kohonen 88, p.154ff]).

As regards the first point, self-organising feature maps perform a similar task as *principal component analysis* or *Karhunen-Loève transform* does[10]. Principal component analysis finds a set of orthogonal vectors in data space that give the direction of highest variance (first principal component), second highest variance in a direction orthogonal to the first principal component (second principal component), and so on. If the input signal distribution is Gaussian, this means max-

---

tions, but it is a slow learner, compared with the Perceptron.

[10]"The map ...will then become aligned along those directions of the signal distribution where the variance is most significant. Moreover, the map attains a format which conforms to the signal distribution" ([Kohonen 82a])

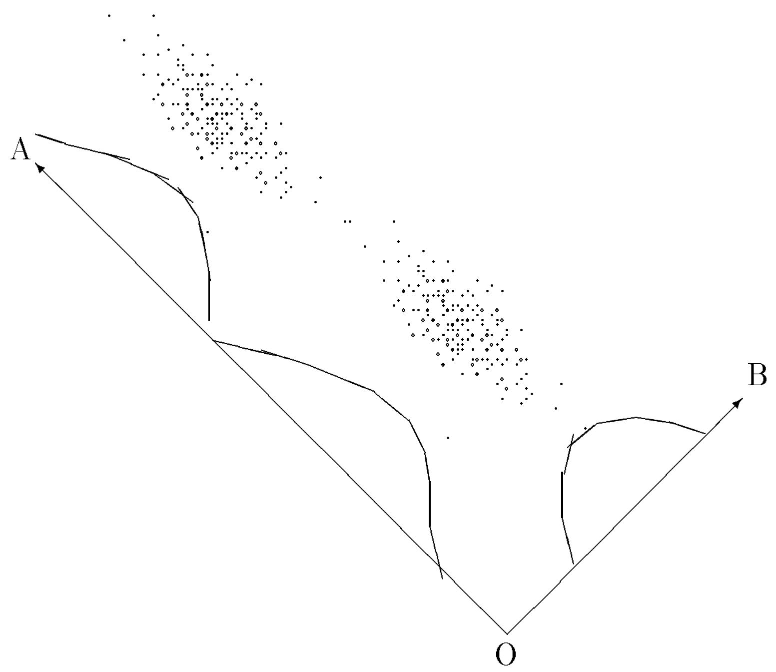

Figure 2.4: Principal Component Analysis.

imising the information content of the output signal ([Hertz *et al.* 91], [Linsker 88]). In figure 2.4 this is shown, OA is the first principal component direction, OB that of the second principal component (after [Linsker 88]).

The self-organising feature map after Kohonen structures the input signal space along the direction of the first principal component; there are networks that can also extract higher principal components from the input data ([Hertz *et al.* 91, p.206]).

Secondly, by mapping a high dimensional input space onto an output space which is ususally one or two-dimensional ([Hertz *et al.* 91, p.236]) self-organising feature maps perform a dimensionality reduction. These properties are exploited in the experiments in location recognition with Alder and Cairngorm: self-organising feature maps are used to cluster the input space (the robots' sensor signals or motor actions, depending on the experiment), retaining the statistical information contained in the input signals.

Such topological mappings can be found in biology, too, for example in the somatosensory cortex ([Churchland 86, pp. 127–130]) or in the visual cortex ([Willshaw & v.d.Malsburg 76]). The fact that the occurrence probability of an input stimulus is encoded in the self-organising feature map can also be found in biology: body areas with high sensor density are mapped onto larger cortex areas ([Churchland 86, p. 129]).

The functionality of self-organising feature maps after Kohonen ([Kohonen 88, ch.5]) is as follows: The input vector $\vec{\imath}$ is the same for all the cells. The output o_j of cell j is determined by the scalar product of input vector $\vec{\imath}$ and the weight vector $\vec{w_j}$ of cell j, and is given by:

$$o_j = \vec{w_j} \cdot \vec{\imath}, \tag{2.4}$$

where $\vec{w_j}$ is the individual weight vector of cell j. The weight vectors are unit vectors: $||\vec{w_j}|| = 1$.

Initially, the elements of the weight vectors are typically set to randomly selected, evenly distributed values ([Kohonen 82a]). Weight vectors are normalised, because the only information used in the self-organising algorithm is information about the spatial orientation of input and weight vectors, not their magnitude. Because weight vectors are randomly initialised one cell of the network is bound to respond most strongly to a particular input presented to the net. The weight vector of this maximally responding cell, as well as the weight vectors of all the cells within a defined neighbourhood around this cell, are then 'turned' towards that particular input vector. This is done by applying the following equation:

$$\vec{w_j}(t+1) = \vec{w_j}(t) + \eta(t)(\vec{\imath} - \vec{w_j}(t)), \tag{2.5}$$

where $\eta(t)$ is the so-called 'gain', a value that determines the amount of turning towards the input vector ($0 < \eta < 1$). η can be chosen to be constant over time or time dependent. In the latter case η is chosen large in the beginning, and small later on to produce bigger changes when the network is newly intialised and small changes when the network is approaching a stable state. After updating, the weight vectors are renormalised. Weight vectors outside the specified neighbourhood usually remain unchanged, in some schemes they are turned away from the input vector (using a suitable function for that purpose, for example a mexican-hat function). The neighbourhood size is often chosen to be decreasing over time, in order to achieve bigger changes in the beginning, and only little changes later.

After several 'epochs', i.e. presentations of input vectors to the network, typical dissimilar responses appear for dissimilar input vectors.

How much 'several' means cannot easily be answered in the case of Kohonen's self-organising feature map; [Hertz *et al.* 91, p. 242ff] and [Ritter 88, p.40ff] discuss the convergence properties of the map in detail. In the experiments discussed in this thesis, the robots could use the SOFM effectively for location recognition after three to four completed circuits round their enclosure (see figure 5.12), which means the presentation of twenty-four to thirty-two input vectors. At that stage the map is not yet completely stable, but stable enough to be used for differentiating between different locations.

2.5 Coping with Abnormal Situations

Talking about a robot's ability to adjust to changing circumstances, two kinds of situation have to be differentiated: that of an undesired situation, and that of an unforeseen one. An undesired situation is one that had been identified by the designer before it occurred: the robot is not in the state it ought to be in (not on the prescribed path, battery charge too low, or a similar situation), but the situation was somehow anticipated. An unforeseen situation, on the other hand, is unexpected and has not been identified before.

Here is an example of an unforeseen situation: Once Alder was filmed by a television team to produce a film for School TV. I was asked to let Alder move into a dead end after the robot had successfully learned to avoid obstacles (i.e. the robot turned right when its left whisker was touched and left when the right whisker was touched). I had never done this before, nor — strangely enough, because it seems such an obvious idea — had it ever even occurred to me before. As far as the designer of the robot controller and the controller itself were concerned, it was an unforeseen situation. Alder went into the dead end, and got stuck. After several unsuccessful attempts to satisfy all instinct-rules, the robot found the exit and left the dead end. Surprisingly, the robot's behaviour had changed: instead of turning to the opposite side of a signalling whisker, the robot turned in one and the same direction (left, in that case) whenever a whisker was touched, regardless of which whisker it was. Thinking about it, this behaviour is better suited to escape dead ends than the obstacle avoidance behaviour (see also section 4.3).

Coping with undesired situations is easier than dealing with unforeseen ones, because a "recipe book" can be supplied by the designer and used by the robot to recover and return to the desired state of operation. [Penders 89] gives an example of such an approach. He defines two classes of errors, unmanageable and manageable[11]. This latter class contains the systematic errors of the robot, which are due to inaccuracies. If such errors occur, a predefined recovery plan is invoked: manageable errors "are those errors for which a recovery plan could be formulated (by us) in terms of the behaviour of the atomic

[11] Presumably the unmanageable errors lead to a halt of the robot, Penders does not state this.

machines."

[Spreng 91] addresses the problem of undesired contacts of a robot arm following a predescribed path. Again, a recipe book is used to recover from such a contact. In what Spreng calls *situation analysis* first a contact hypothesis is formulated (i.e. what sort of contact could have occurred?). In the next step, a suitable test motion is determined in order to verify or refute the hypothesis, the test motion is then executed and, in the fourth and last step, evaluated. The result of the evaluation determines the choice of an appropriate recovery action. Using recipes for error recovery improves robot performance within known boundaries, it allows robots to cope with foreseen errors due to known inaccuracies of the robot. To move towards the stated goal of this thesis — flexibility in unforeseen situations — this method is unsuitable, simply because the recipes are not known.

Dealing with robot inaccuracy can be done either by applying predefined error-recovery recipes (see the two examples above), or by using robot behaviours that are less prone to error. The *SOMASS* system ([Smithers & Malcolm 87, Malcolm & Smithers 88]) is an example of the latter. Without locating parts through computer vision, tactile sensing or other kinds of sensing, an Adept assembly robot is able to pick up parts that have been put by a person in a particular location (this means that the parts are more or less where the robot expects them, but not in a precisely known place). How is it done? The robot first picks up a brush, brushes along the table in a defined manner, which will leave the part in a precisely defined location, whatever its original place (within reason) was. Then the robot picks it up. In the experiments discussed here, I have also tried to use behaviours that are not very prone to error. In mapbuilding, for example, the robots follow the wall of their enclosure, instead of moving arbitrarily. The wall following behaviour is robust, because it basically consists of two simple actions: turn to your right (to see whether the wall is still there), and turn back (to be parallel to the wall again). If the robot relied on travelling across the enclosure at a particular angle, it would fail often, due to the inherent variation in robot and world.

[Simon *et al.* 90] use self-tuning of motion-level robot program primitives to cope with variation inherent in the world. In their approach, bounded value parameters are adjusted to minimise a cost function defined by the designer.

On the engineering side, adaptive control is related to the problem discussed in this section. Industrial controllers (usually PI or PID controllers) are used widely for clearly specified control tasks. Because tuning these controllers is difficult and time consuming, they are often badly tuned. To overcome this problem, adaptive controllers which could adapt to the control problem were introduced in the early 1980s. In 1988 about 100,000 adaptive loops were running ([Åström 89]). Åström sees the following difficulties in installing adaptive controllers:

> "To work well, the algorithms require prior information about the sampling period, dead time, model structure, and signal ranges. Knowledge about time scales is particularly important...Control theory has been very successful in giving exact answers to precisely stated problems. However, much less work has been devoted to simple methods that give order-of-magnitude estimates. There has been a strong emphasis on linear problems, whereas many practically important nonlinear issues have been neglected."

This means that if the control problem is understood and if sufficient prior information is available, adaptive controllers can be well used. However, because such information concerning *unforeseen* situations is not available, adaptive controllers were not used on Alder and Cairngorm.

2.6 Summary

Alder's and Cairngorm's tasks can be divided into four main areas: staying operational, navigating, learning and coping with abnormal situations.

Staying operational for a mobile robots means to avoid obstacles, to get unwedged if trapped and to avoid potentially hazardous areas. Two main approaches can be seen: the analytical approach analyses the whole control task and implements it through a series of functional units; this is a top down approach. The synthetical approach, on the other hand, achieves the overall control task through parallel operation of independent modules (the global behaviour of the robot is emergent from the simultaneous operation of these modules); this is a bottom up approach. Examples of both both approaches and a detailed discussion have been given in section 2.2.

Navigation comprises two main tasks: knowing the current position (location recognition), and knowing how to get to a goal position (map interpretation). In order to be able to navigate, a robot needs some sort of internal representation of its environment. Geometrical maps contain information about distances and angles between locations, and therefore also information about the neighbourhood relationships between locations. Such geometrical maps are usually predefined by the designer and then, in many cases, updated during runtime of the robot, using sensor readings. Topological maps contain information only about neighbourhood relationships between locations, for this reason they are less prone to noise and variation inherent in sensors, actuators and the world. Examples of both approaches, implemented in robots, are given in section 2.3.

If the appropriate behaviour of a robot is initially unknown, or if it is desired to enable the robot to adjust to changing circumstances, learning controllers can be used. To date, in particular reinforcement learning (learning by trial and error through performance feedback) has been used both on real robots and in computer based simulations. In Alder's and Cairngorm's case reinforcement learning is implemented using a connectionist computing architecture. A discussion can be found in section 2.4.

If robots are to be used in changing environments and if they are to stay operational even under the influence of noise and variation, the ability to cope with abnormal situations is necessary. Abnormal situations are both undesired situations — these have been identified beforehand and are therefore not unexpected — and unforeseen situations, which have not been identified before. In order to cope with undesired situations, predefined strategies, self-tuning controllers or adaptive controllers can be used; however these methods fail if unforeseen situations occur. Alder and Cairngorm use their learning capability to cope with the latter. A discussion can be found in section 2.5.

Chapter 3

Reality or Simulation

> The length of your legs makes a great difference to your life.
>
> David McFarland

3.1 Introduction

In section 1.1 I have given the example of two questions being asked, which, although being identical in contents, received completely different answers (the example of the Asian disease). The actual wording, not the subject matter of the question determined the outcome of the survey.

Similarly in robotics the research methodology chosen, and particularly the tool used for investigation will influence the final results and their validity. It is therefore a question of importance; I address it in this chapter.

In general, a research methodology comprises three aspects:

1. a theory, for example the laws of physics, mechanics, thermodynamics and material sciences,

2. a tool, for example that of computer based simulation, and

3. an evaluation, for example that of comparision of simulation results with experimental results.

Whereas laws of physics, mechanics, electrical engineering, mathematics and related sciences, as well as biology, ethology and neuroscience are widely accepted as (some) underlying theories in robotics research[1], practices differ with regard to the tools used for investigations.

The majority of workers in the area of intelligent agents/intelligent mobile robots to date use computer based simulations as tools for their investigations (many examples for this fact can be found, for instance, in [SAB 91]). This fact indicates that there are advantages to this approach, as compared to performing actual experiments. The most important reason is, perhaps, that the equipment needed for computer based simulation (the computer) is available in every AI laboratory and in every department of Computer Science, whereas the equipment needed for experiments (most of all, a robot), is not. However, there are other reasons which have to do with the fact that dealing with the real world presents problems that can be bypassed using simulation. [Tyrrell & Mayhew 91, p.263f]:

> "One approach to examining behaviour has involved the building of robots which can navigate in a selected environment and which are given a limited behavioural repertoire. Research of this kind not surprisingly requires a large amount of time to be spent on perception and motor control. Behavioural strategies for a robot cannot be tested if the robot is not able to properly sense its environment or if it is not able to put its chosen behaviour into effect. We decided that using a simulated environment would allow us to bypass the sensory perception problem (we just provide the animal with the relevant information about the environment), and also to bypass the motor control problem (we just calculate the effects of the animal's

[1] Complete theories however, as to what constitutes relevant parameters in robot behaviour are as yet very brittle if not missing at all.

behaviours on the simulated environment without concerning ourselves as to how they are achieved). This would allow us to concentrate on behavioural issues."

The three main claims given here are that

1. simulation is quicker than experimentation;

2. simulation allows to bypass the sensory perception problem (noisy sensors; sensor interpretation in general) and the motor control problem (inaccurate actuators); and

3. simulation is easier to construct because "the relevant information" can be provided directly and does not have to be deduced from noisy sensor data.

[Kaelbling 90, p. 171] supports the first point: "The ... problem [of conducting experiments with robots] is that it takes a long time to conduct the experiments. ... So, instead of trials on the real robot, we must substitute a simulation of the robot and its domain.".

I do *not* believe that simulation is generally quicker than experimentation. Surely Tyrrell and Mayhew are not speaking about overly simplistic simulations, but about accurate and faithful models of a real world environment, and models of robot sensors and actuators that are faithful to their physical originals. If simulation is to say anything about robot behaviour, this is the only possible way. If a simulation is of low fidelity, it " causes this to be a substantially different problem than that of running on the actual robot."([Kaelbling 90, p.171]). To build such a simulation of high fidelity, however, is not trivial and requires at least as much time as building a small robot ([Donnett & Smithers 91, Martin *et al.* 90, Nehmzow *et al.* 89]) and letting it interact with the real world.

The other points raised by Tyrrell and Mayhew ("concentrating on behavioural issues by bypassing the sensory perception problem

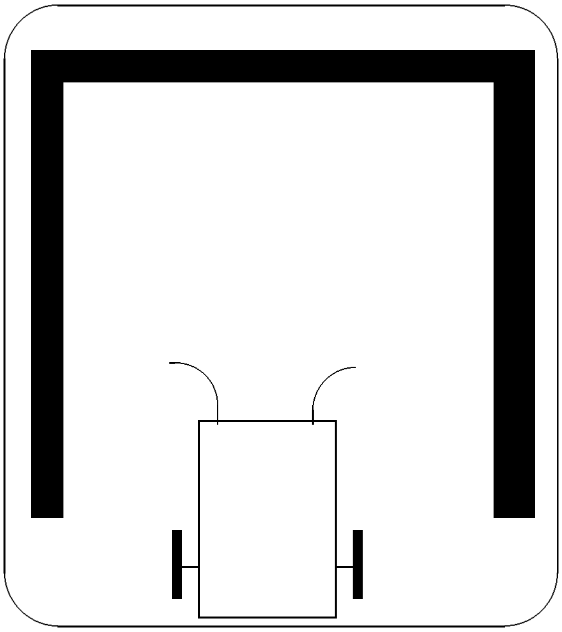

Figure 3.1: A robot in a dead end.

and bypassing the motor control problem"), however, make computer based simulation seem an attractive tool for the investigation of robot behaviour, and for this reason I will take a closer look at this tool in order to find out whether it is suitable for the research outlined in section 1.1. In the next section I will therefore present an experiment in simulation.

3.2 The Dead End Example

The task I addressed in both a simulation and an experiment was that of a mobile robot with two tactile sensors at the front and two motors, encountering a dead end (see figures 3.1 and 3.2). This is a simple enough task and it should be relatively straightforward to obtain similar results through simulation and experiment.

First I simulated this particular situation, using a simulator which

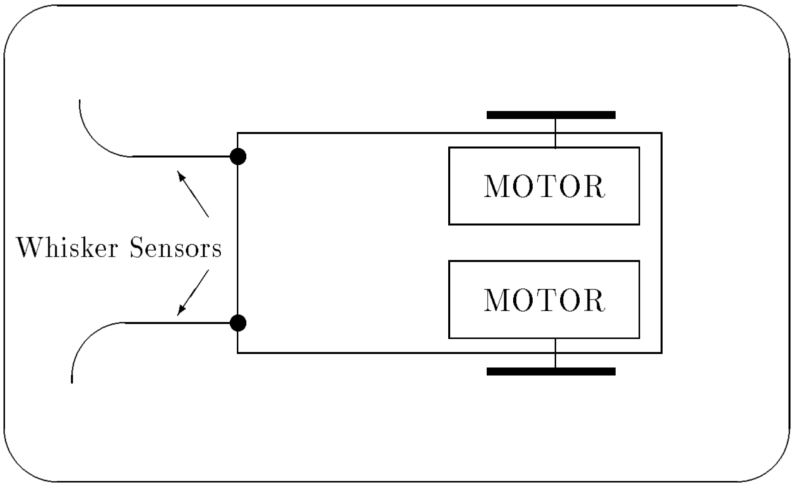

Figure 3.2: The morphology of the simulated and the real robot.

had originally been written to imitate Alder's behaviour. Then I replicated the behaviour of the simulated agent on a robot, on Alder. The idea, of course, was to confirm that both simulation and experiment would lead to similar results.

3.2.1 The Simulation

The simulator was written by members of the "Forann Project" at Bremen University's Department of Computer Science, with whose kind permission I used it. Following [Nehmzow *et al.* 89] and [Nehmzow *et al.* 90], they copied the basic behavioural patterns of Alder to the simulation, which I call Driesh[2]. The simulated agent has three whisker sensors, mounted at its front, and is able to move forward and rotate on the spot. Like Alder, it is also able to learn through trial and error and acquire competences such as obstacle avoidance and wall following. Figure 3.3 shows a typical scene: Driesh has found a wall

[2]Driesh is a mountain in the Scottish Highlands, like Ben Alder and Cairngorm.

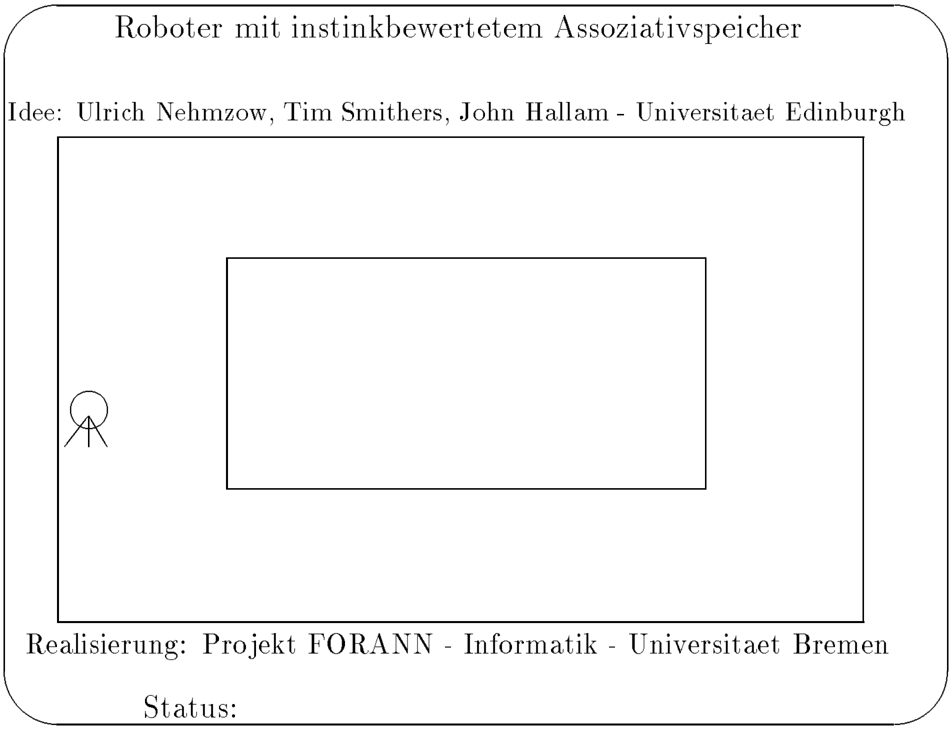

Figure 3.3: Driesh: Alder — simulated.

and is following it[3].

Here I am not concerned with the simulation of competence acquisition, that is the simulation of the actual learning algorithm; instead I used the simulation to find out about similarities and differences between Alder and Driesh. In order to understand the configuration of both simulation and experiment, I briefly have to explain how Alder and Driesh learn from experience. The whole learning process is explained in detail in chapter 4.

When Alder encounters a situation in which any of the robot's instinct-rules[4] are violated, it tries to perform a motor action that results in the violated instinct-rule being satisfied again. Initially, when

[3]This figure is redrawn from an original screen image.

[4]Constant rules that are used to assess the robot's performance.

no previous knowledge is available, the robot will perform the motor action associated with the first output node of the artificial neural network that controls the motor actions. This is a left turn. If this left turn does not yield the desired result (the instinct-rule being satisfied again), the robot will try the next move, a right turn, *for a little longer*. If that does not satisfy the instinct-rule, the robot will try the next choice (in the simulation and in most experiments another left turn), and again a little longer than the previous right turn. Eventually the robot finds a motor action that yields a satisfied instinct-rule, and will associate that motor action with the initial input stimulus.

In order to compare the behaviour of a simulation with that of a robot, an interesting experiment would be to let the simulation and the actual robot perform left and right turns in order, as just described, but *not* to prolong the execution period each time, and make both simulation and robot negotiate a dead end (as shown in figure 3.1). Figure 3.4 shows the parameter setting to simulate this. The parameter value of "ucmult"[5] is set to one, which will make Driesh prolong the trial period of each action by factor 1 (i.e., keep the trial period constant), whenever a new action is initiated. By the way, this way to extend the durations of motor actions (multiplication) is unlike the method used on Alder (Alder and Cairngorm use an additive method), but for this example it does not make a difference, because there is to be no extension of duration. Figure 3.4 also shows the definition of the simulated environment, initial position, obstacles etc.

In short, this simulation is configured to keep the simulated whiskers quiet, and if they do issue a signal, to try left and right turns of equal and constant duration in sequence in order to move away from the obstacle.

[5] "ucmult" determines by which factor trial periods of motor actions are extended if the previously executed action was unsuccessful.

```
# Number of border points
borderpoints 4

# Number of obstacles
obstaclenum 2

# Driesh's starting location
robotstart 0.3, 0.02

#  Compass heading in degrees
heading 90.0

# Length of one step, must be smaller than sensorlenght/2
movestep 0.0029

# Turning angle in degrees
turnstep 6.0

# Length of a sensor. Must be greater than 2 x movestep
sensorlength 0.030

# Angle between middle sensor and outer sensors in degrees
sensorangle 35.0

# Number of iterations before "touch-something" triggers
touchtime 4

# Initialise intial trial period of a motor action
ucinit 4

# Factor to increment trial period if unsuccessful
ucmult 1

# If gdebug=1, path of Driesh is shown graphically
gdebug 1

# If trace 0, no trace
trace 0

# Simulated noise lies between 1 und 1 + [0,1[ / Fuzzy
fuzzy 10.0

# Definition of border points
borderstart:
0.0 , 0.0
1.0 , 0.0
1.0 , 1.0
0.0 , 1.0
borderend:

# Definition of obstacles
obstaclestart:
# Number of obstacles
obstaclepoints 2
0.2 , 0.051
0.5 , 0.051
obstaclepointsend:
obstacleend:

# End of configuration
end
```

Figure 3.4: Simulation parameters used for dead end experiment.

Figure 3.5 shows the outcome of the simulation[6]. Driesh has moved forward into the dead end to the facing wall. There it oscillates between the left and the right wall, forever. Even though the turning angles are subject to artificial noise, this oscillation went on for over ten minutes, the simulation was then manually terminated.

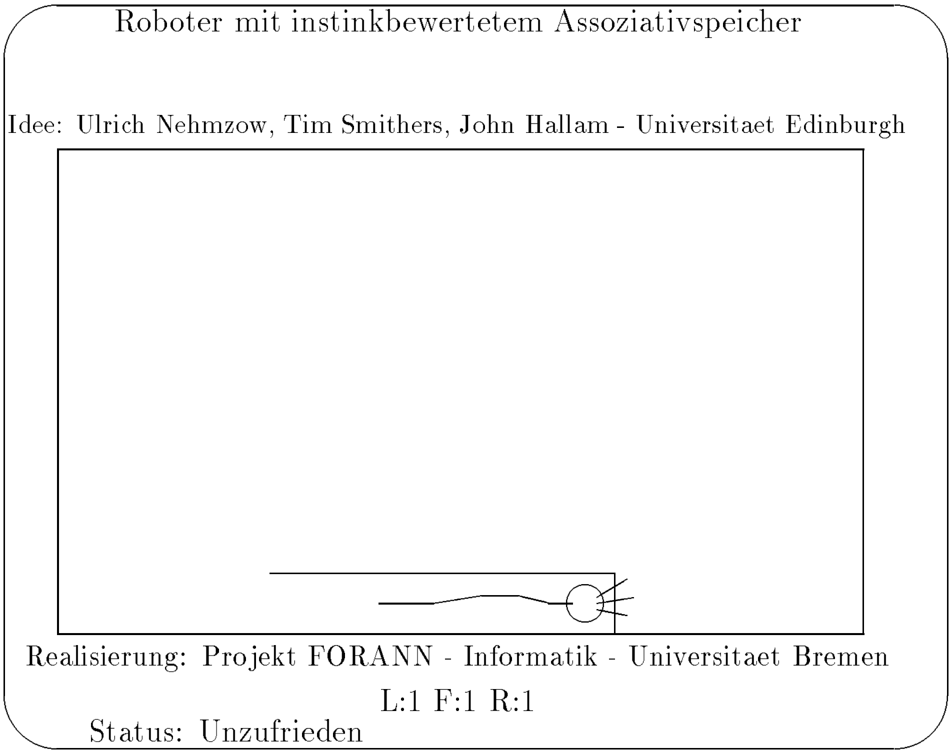

Figure 3.5: Simulation of the dead end experiment.

3.2.2 The Experiment

The short program shown in figure 3.6 intends to replicate Driesh' behaviour on Alder.

As in the simulation, the robot is programmed to move forward if no sensor signals are received, and to turn left and right for equal and

[6] Again, this figure is redrawn from an original screen image.

```
REM THIS IS THE EQUIVALENT CONTROL PROGRAM TO THE SIMULATION
REM CONCERNING  DEAD END ESCAPE

1 TURNL=6
2 TURNR=9
3 AHEAD=10
5 HALT=0
6 NOSIGNAL=0
7 SWITCH=8

10 XBY(7000H)=HALT
15 GOSUB 5000:PRINT SENSOR
20 IF SENSOR<> SWITCH THEN GOTO 15

30 GOSUB 5000
40 IF SENSOR=SWITCH    THEN GOTO 10
50 IF SENSOR=NOSIGNAL  THEN XBY(7000H)=AHEAD:GOTO 30

REM PERFORM LEFT TURN FOR A WHILE
60 IF SENSOR<>NOSIGNAL THEN XBY(7000H)=TURNL
70 WAIT=0
80 GOSUB 5000

90 IF SENSOR=NOSIGNAL THEN GOTO 50
100 WAIT=WAIT+1
110 IF WAIT<30 THEN GOTO 80

REM PERFORM RIGHT TURN FOR A WHILE
120 XBY(7000H)=TURNR
130 WAIT=0
140 GOSUB 5000
150 IF SENSOR=NOSIGNAL THEN GOTO 50
160 WAIT=WAIT+1
170 IF WAIT<30 THEN GOTO 140

180 GOTO 60


REM READING THE SENSOR, IGNORING BOUNCING
5000   SENSOR=XBY(6C00H)
5010 FOR CC=1 TO 5
5020 IF XBY(6C00H)=0 THEN SENSOR=0
5030 NEXT CC
5040 RETURN

6000 END
```

Figure 3.6: Alder's control program for the dead end experiment.

constant durations of time as soon as any of the whiskers gives a signal. From the simulation conducted earlier one would expect the robot to remain in the dead end and oscillate, just as Driesh did.

What actually happened is shown in figure 3.7: very quickly the robot leaves the dead end!

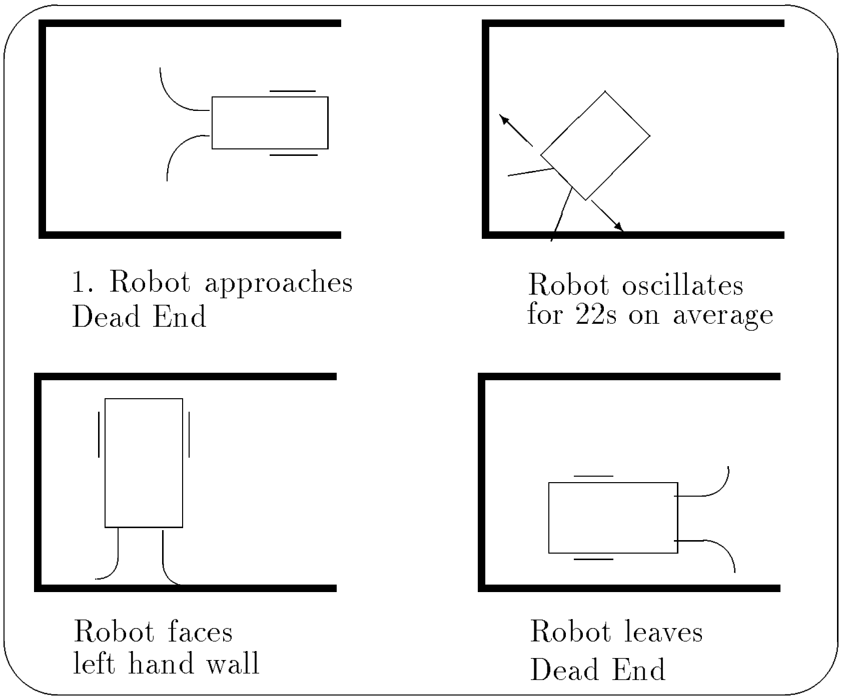

Figure 3.7: Cartoon of dead end experiment.

This behaviour is not a one-off event, in ten out of ten experiments the robot left the dead end, the times this took in each case are shown in figure 3.8.

Trial No.	1	2	3	4	5	6	7	8	9	10
Time to escape	28s	14s	43s	17s	9s	26s	56s	8s	9s	13s

Figure 3.8: Time needed for Alder to escape from dead end

Following page: photograph of the dead end experiment (explanation on page 67).

The traces of three of these ten experiments are visible in the photograph after page 66, in the experiments marked 1 and 2 Alder left the dead end almost immediately, in experiment 3 it took a little longer (the thick semi circle is the result of many left and right turns).

3.2.3 Discussion

Although the experiment tried to replicate the behaviour observed in the simulation, it failed to do so. The control programs of robot and simulation were as equivalent as I could make them, and indeed, when encountering a single, convex obstacle both robot and simulation do behave similarly. Yet in the dead end context parameters not represented in the simulation, but implicitly present in the robot and the world lead to radically different results. In this case, the fact that Alder does not turn symmetrically, largely due to differences in motors and gearboxes, makes the difference: each time the robot turns a little more left than it turns back right, so that eventually it faces the left wall, and then leaves the dead end. In general real robots tend to benefit from the fact that noise and stochastic processes are present in the world, they often allow the robot to recover in situations where simulated agents are trapped because of more deterministically defined environments.

This shows how difficult it is to "just provide the relevant information about the environment ... and to just calculate the effects of the agent's behaviour on the environment". To get this modelling right is the crucial part of any simulation; if the relevant parameters of world, task and agent are not identified, the simulation is of very little use. Overly simplistic simulations do not say much about robot behaviour. This is the fundamental problem of any simulation, that the core of it relies on the designer's ability to identifiy relevant features in the original behaviour that is to be modelled. In cases like the turbine example this is possible, because the underlying laws are largely understood. In cases of intelligent mobile robots the intelligence of the robot is to a great extent not in the robot, nor in the world, but arises through the interaction between robot and environment. I have said earlier that the interaction of individual, simple components can produce effects that are not present in the mere collection of them (see figure 1.1). This can be found in biology (e.g., mound building of ants) as well as in artificial systems. Steels, for example, presents a simulation in which intelligent overall behaviour is achieved through the interaction of simple agents with a simple environment ([Steels 89]). Similar considerations apply to robotics, the factors governing this process are *not* yet clearly understood (see also [Brooks 91a]). From this I drew the conclusion that in order to conduct research in robot behaviour as outlined in section 1.1, the appropriate means to do this would be to conduct experiments with robots[7].

However, there is a place for faithful simulation. Simulation is a top down approach to analysis of behaviour: the relevant parameters in agent, task and environment are analysed *a priori*, modelled and implemented in the simulation (similar to the analytical approach to robot control, mentioned earlier). On the other hand, the experiment is a bottom up approach to the analysis of behaviour: the behaviour is produced first, in the desired environment, by the agent performing the desired task. Then the relevant parameters are analysed, i.e., a theory is formed. These two approaches can meet in the middle: a postulated theory, derived from experiments, can be evaluated through a simulation if sufficient data from experiments is available. In gen-

[7]Talking about mobile robotics, Rodney Brooks came to the same conclusion: "Don't use simulation as your primary testbed. In the long run you will be wasting your time and your sponsor's money" ([Brooks 87]).

eral, computer based simulation is a useful tool if sufficient theory and experimental data are available (see also figure 3.9).

	Foreseen Situations	Unforeseen Situation
Good Data and Theory	Experiment or Simulation	Experiment
Little Data and Theory	Experiment	Experiment

Figure 3.9: Suitable tools for the investigation of behaviour.

Both the top down and the bottom up approaches have weak points. If the modelling of agent, task and environment is inaccurate, if the wrong components were identified as being important, then the simulation is of little value as a tool for the investigation of behaviour. Experimentation, on the other hand, does not suffer from the problem of over-simplification, because by the nature of it *all* features of both robot and particular environment are there. As [Brooks 91b] puts it:

> "It is no longer possible to argue in conference papers that the simulated perceptual system is realistic, or that problems of uncertainty in action will not be significant. Instead, physical experiments can be done simply and repeatedly. There is no room for cheating (in the sense of self-delusion, not in the sense of wrong doing with intent)."

However, like simulation, the experiment suffers from the problem that the experimenter's own interpretation comes in at some stage, and that it might be wrong. His analysis of an experiment could be wrong, which would mean that the robot would still be doing what it was doing, but the observer would believe this to be happening for the wrong reasons. I believe that because the investigation of robot behaviour is the primary aim, experimentation is at the moment the better choice, as it inevitably results in robot behaviour in the real world.

The methodology I chose, therefore, is this:

- Theory: robot behaviour is determined both by identifiable parameters such as the laws of physics, mechanics, electrical engineering etc., and by parameters yet unidentified, arising from the interaction of agent and environment.
- Tool: experimental investigation, using robots.
- Evaluation: statistical analysis through repeated experiments.

In the next section I describe the two robots I used, Alder and Cairngorm, in more detail. The subsequent chapters then describe the experiments conducted.

3.3 Hardware

Alder was the first of the Really Useful Robots, it is shown after page 74 and after page 18 (left). Alder consists of a chassis built from Fischertechnik (a technical construction kit), an ARC 52 controller which uses an INTEL 8052 eight-bit microprocessor and has an on-board BASIC interpreter (this controller card is clearly visible on pages 74 and 18(left)), an interface card (partly obscured in the photograph after page 74) giving independent control for two motors (forward, reverse, and stop for each motor, but with no feedback of distance travelled or angle turned), and up to eight binary sensor inputs. A schematic diagram of Alder is shown in figure 3.12. For most of the experiments I have used two whisker sensors which act as omnidirectional tactile sensors (see figure 3.10) and push button switches for manual control of the robot base (the whiskers are mounted at the front of the robot and are clearly visible in the photograph after page 74, the pushbutton switches are mounted on the top of the robot).

A revolution counter was used for distance measurement in experiments on navigation (the cam is visible at the back of the robot, it can

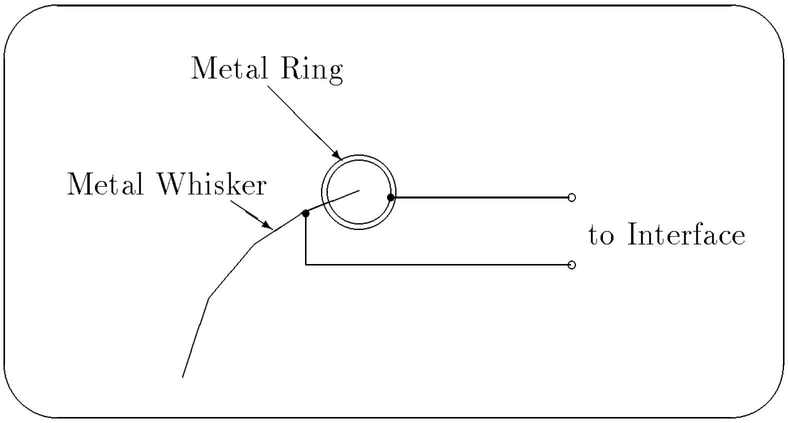

Figure 3.10: Schematic diagram of the whisker sensors used on Alder and Cairngorm.

be used to press a pushbutton switch), and the forward movement sensor was used in experiments on skill acquisition (see figure 4.14). This forward movement sensor (see figure 3.11) is a pushbutton operated by the caster wheel of the robot, it is not shown on the photographs.

For some experiments with Alder a Polaroid ultrasonic range finder was used, the active range of this sensor is from about 20cm to 3m. The sonar sensor is not shown on the photograph.

Cairngorm (see photographs after pages 76 and 18 (right)), the third[8] of the "Really Useful Robots" also uses a chassis built from Fischertechnik, but has a more powerful controller (Flight 68k) based on a Motorola 68000 CPU and having 128 Kbytes of memory (the controller card is clearly shown in both photographs). Its motor control and sensor inputs are similar to that of Alder, but it is programmed in C, rather than BASIC. Otherwise it is essentially the same as Alder. A schematic diagram of Cairngorm is shown in figure 3.13. Up to six binary sensors can be connected to Cairngorm's interface, normally

[8]The second robot in the series is Ben Hope, which was used for a different set of experiments.

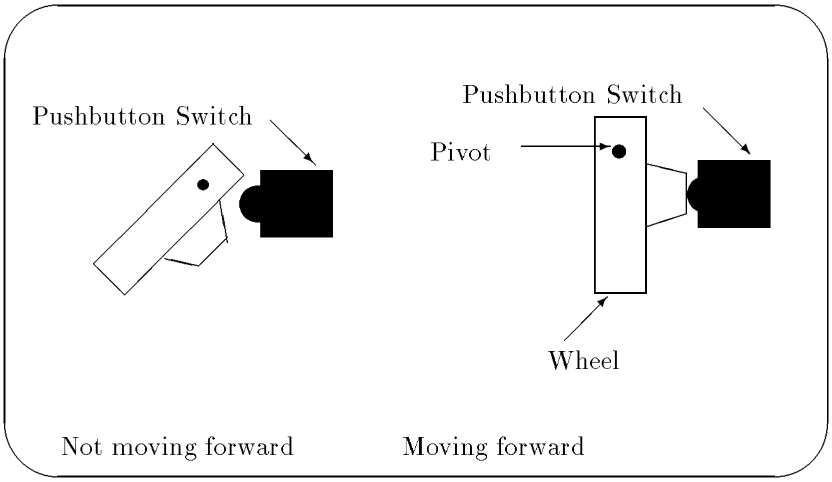

Figure 3.11: The forward motion detector of Alder.

these are tactile sensors and pushbutton switches. I have not built a revolution counter or a forward motion sensor for Cairngorm, but if desired this is possible just as in Alder's case. Cairngorm does not have an ultrasonic range finder.

Both robots run completely autonomously, they solely use their on-board computers for control. At the beginning of an experiment, programs are downloaded via a serial link from a workstation, are started and then executed on the ARC 52 or Flight 68k controller board respectively. It is possible to monitor any messages sent by the robots on the workstation if the link remains connected; this is usually only done during the test phase. For the actual experiments reported in this thesis the robots operated completely autonomously, with neither communication nor power link to a base station.

That sensor type and travelling speed are tightly coupled is a principle that can be found in biology as well as technology. The range of Alder's and Cairngorm's sensors is short, consequently they travel at a low speed. Alder's maximum speed is about 3 cms^{-1}, Cairngorm's maximum speed is about 2 cms^{-1}.

3.4 Summary

In the experiment presented in this chapter I tried to replicate the behaviour of a simulated agent encountering a dead end on a robot. However, this failed because the overall behaviour of a robot is governed not only by factors which are immediately obvious, such as sensor signals, motor actions, dynamical properties etc., but also by seemingly unimportant aspects such as slip, backlash, wear etc., as well as by factors arising from the interaction of robot and world (pushing hard against an obstacle, for example, will either displace the obstacle or the robot, which changes the relative positions of obstacle and robot). Whilst Driesh (the simulated agent) remained stuck in the dead end, Alder (the robot) escaped.

In general, computer based simulation can be a useful tool, provided sufficient theory and experimental data is available. As the research topic of this thesis is the investigation of robot behaviour under unforeseen circumstances — an area where little experimental data is available — I decided to conduct experiments, using robots.

For this purpose I have built two mobile robots, Alder and Cairngorm, which are controlled solely by on-board computers and which can operate completely autonomously. These robots are shown on page 18.

Following page: photograph of Alder.

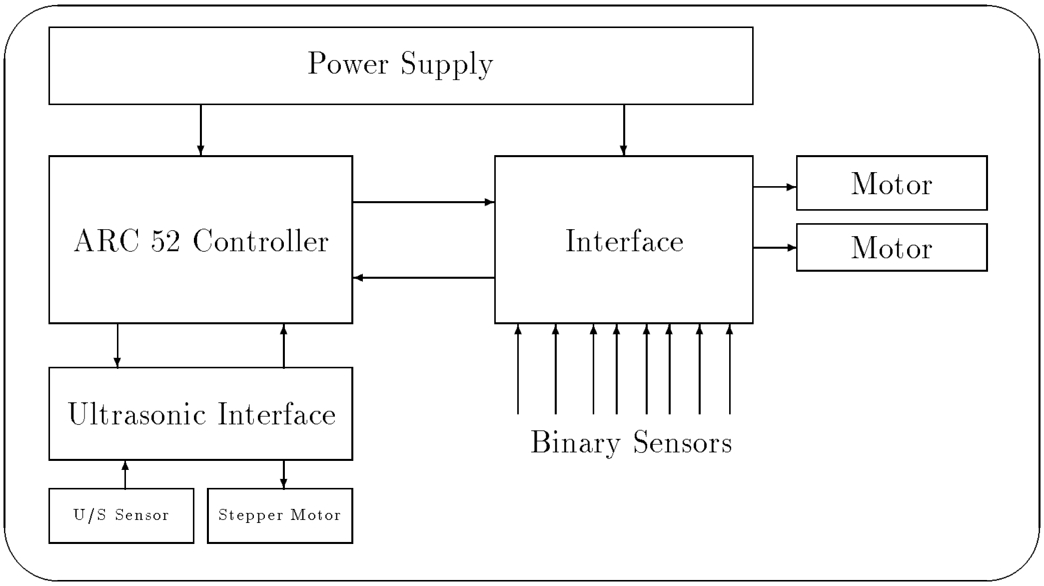

Figure 3.12: Schematic diagram of Alder.

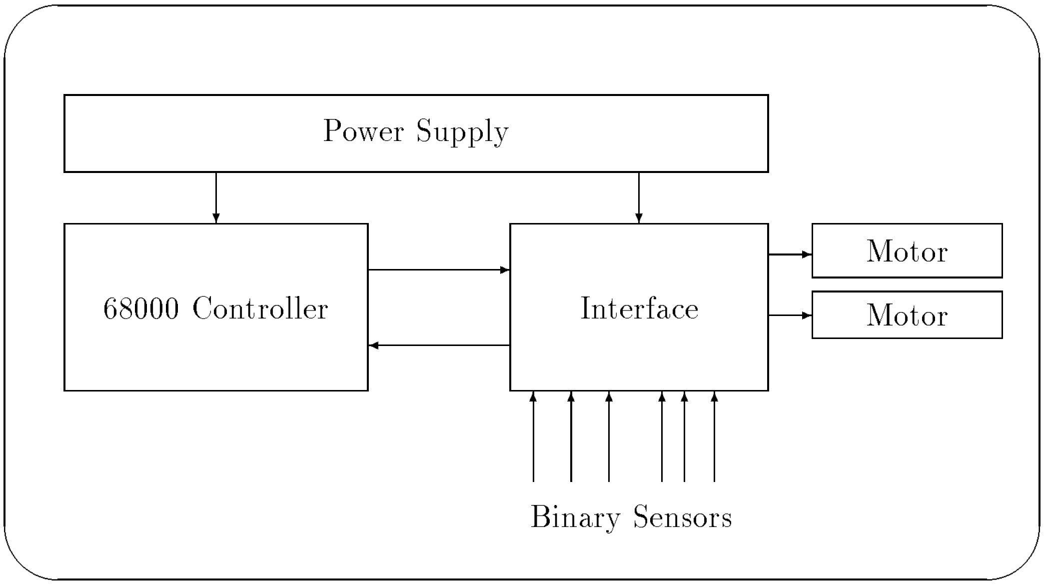

Figure 3.13: Schematic diagram of Cairngorm.

Following page: photograph of Cairngorm.

Chapter 4

Skill Acquisition using Artificial Neural Networks

4.1 Introduction

Having looked at the current state of research in intelligent mobile robotics and having discussed the reasons for doing robotics rather than simulation I will now, in the following chapters, describe the experiments conducted with Alder and Cairngorm. This chapter describes the experiments I conducted in motor-sensory competence acquisition, in which the robots autonomously determine the effective wiring between their sensors and their actuators in order to achieve obstacle avoidance, wall following, corridor following etc. Chapter 5 describes experiments concerning navigation, and chapter 6 links and combines the experiments described in this chapter and chapter 5.

4.1.1 The Controller Architecture

As mentioned in section 1.1, the goal of the experiments described in this thesis was to achieve flexibility in unforeseen situations in mobile robot control. This, I believe, requires the robot's ability to learn from experience. I therefore chose to build a controller with a learning capability, so that the robot could determine for itself an effective wiring between sensors and actuators. In Alder's and Cairngorm's controllers these associations between sensors (input stimuli) and actuators (responses) are stored in a Pattern Associator, as described in section 2.4.2.

Figure 4.1 shows the general structure of the controller used in all experiments discussed in this chapter. The controller consists of fixed and plastic components, fixed components being the instinct-rules, the robot morphology and various parameters within the controller; the plastic component is the Pattern Associator.

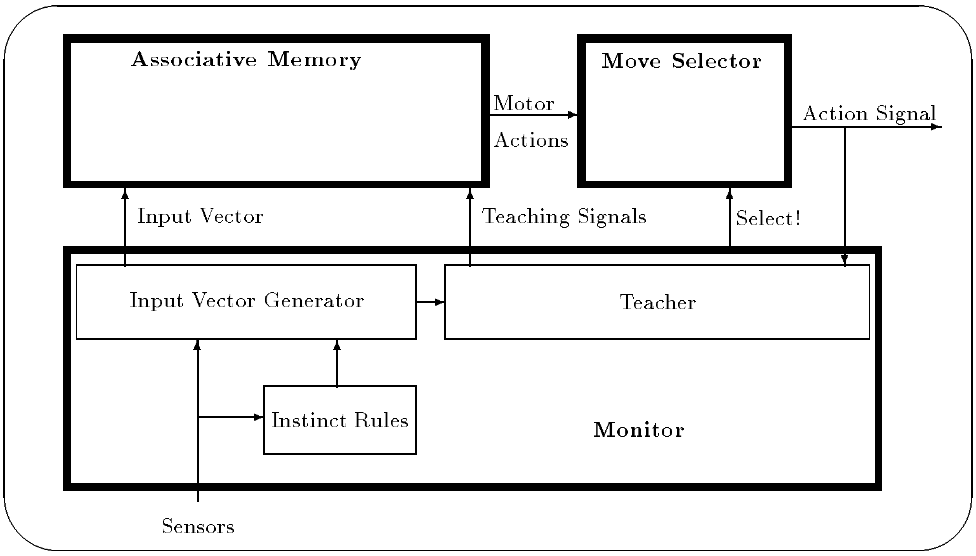

Figure 4.1: Computational Structure of the self-organising Controller.

4.1.2 Instinct-Rules

As the Pattern Associator is trained by 'supervised learning', a means of evaluating the robot's behaviour and determining teaching signals for the Pattern Associator had to be found. I use fixed rules for this purpose which I call *instinct-rules*. They are similar, but not identical to *instincts* as defined by [Webster 81]: an instinct is a "complex and specific response on the part of an organism to environmental stimuli that is largely hereditary and unalterable though the pattern through which it is expressed may be modified by learning, that does not involve reason, and that has as its goal the removal of a somatic tension or excitation". This describes behaviour and is therefore different to the instinct-rules used in the experiments decribed here, as instinct-rules are not behaviour, but constants that guide the learning of behaviour. The goal of instinct and instinct-rules, however, is the same: the removal of a somatic tension (in the case of the robot such a 'somatic tension' is an external sensor stimulus, or, in some experiments, the lack of it).

Each instinct-rule has a dedicated sensor in order that it can be established whether it is violated or not. This sensor can be a physical, external sensor (for example a whisker), or an internal sensor (for example a clock that is reset every time some external sensor stimulus is received). Even memory can serve as a trigger for an instinct-rule (Touch alternate walls!).

4.1.3 Input and Output

Current and previous sensor signals constitute the input signals to the Pattern Associator[1].

The output of the network denotes motor actions of the robot. The networks used in the self-organising controller presented here have four output nodes, in most cases denoting just left and right turn (twice), however if required four different motor actions (including forward and

[1] Information about violated instinct-rules could similarly be used; but this has not been done in the experiments described in this thesis.

backward movement as well) can be assigned to the four output units[2]. The reason for the restriction to two possible motor actions is that for tasks such as obstacle avoidance and wall following, left and right turn are the only motor actions needed. Too many output nodes (i.e., possible motor actions) can cause the robot to learn functions that, although satisfying the instinct-rules, result in undesired behaviour. In early experiments it has happened that the robot acquired an obstacle avoidance behaviour in which it oscillated in front of a wall, moving forward and backward (this also satisfies the instinct-rules shown in figure 4.11). One way to prevent this was to assign left and right turns to the first two output nodes of the network (in an untrained network these are tried first, because initially all weights of the network are zero), but it turned out to be more effective to restrict the robot's repertoire of motor actions to left and right turns in the first place.

The idea behind this controller setup is that effective associations between sensor signals and motor actions arise over time through the robot's interaction with the environment.

4.1.4 Mechanism

As said earlier, the Pattern Associator requires a teaching signal to develop meaningful associations between its input and its output. This teaching signal is provided by the *monitor*, a "teacher" that uses the instinct rules to assess the robot's performance and teach the network accordingly: as soon as any of the instinct-rules become violated, an input signal is generated by the *input vector generator*, sent to the associative memory and the output of the network is computed. The *move selector* determines which output node carries the highest out-

[2]That I used four output nodes for two motor actions has historic reasons: the first controller built used all four motor actions.

put value and which motor action this output node stands for. That motor action is then performed for a fixed period of time. For Alder and Cairngorm a typical value is about four seconds. If the violated instinct-rule becomes satisfied within this period of time, the association between original input signal and output signal within the Pattern Associator is taken to be correct and is confirmed to the network (this is done by the monitor). If, on the other hand, the instinct-rule remains violated, a signal is given from the monitor to the move selector to activate the motor action that is associated with the second strongest output node. This action is then performed for a slightly longer period of time than the first one (in Alder's and Cairngorm's case the increase is typically two seconds) to compensate the action taken earlier; if this motor action leads to satisfaction of the violated instinct-rule the network will be taught to associate the initial sensor state with this type of motor action; if not the move selector will activate the next strongest output node. This process continues until a successful move is found.

The fact that motor actions are tried in turn and with increasing duration is important for making this skill acquisition scheme work.

If, to give an example, the robot had two possible motor actions available (left and right turn) and was trying to touch a wall, due to a built-in boredom instinct-rule being triggered, it would initially turn left and perform this action for four seconds. Supposing the wall *was* on the left of the robot, but the robot did not touch it within the first four seconds, the right turn would be initiated and executed for six seconds. This would leave the robot pointing a further "two-seconds-worth" away from the wall. After these six seconds, the robot would again move to the left, this time for eight seconds, and touch the wall. This will lead to the association of a left turn with the boredom instinct-rule, which happens to be the correct association. So, although the very first left movement of the robot was unsuccessful, the robot

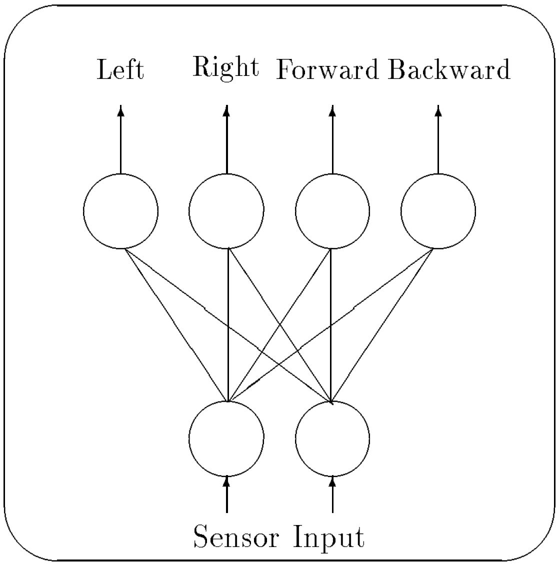

Figure 4.2: Association between sensor stimuli and motor actions.

nevertheless was able to discover the correct motor action.

Figure 4.2 shows the general structure of the Pattern Associator used. The actual input to the network may vary from experiment to experiment (the input vectors are described at the relevant places in this thesis), the output nodes denote motor actions (the figure shows the more general case with four possible motor actions; usually a network with just two motor actions — left and right turn — is used).

The result of this process is that effective associations between input stimuli and output signals (motor actions) arise. Obviously, this is only possible with the right sort of fixed parameters, in particular the instinct-rules play an important role. The instinct-rules and sensors I used are described in more detail in the following sections, where I will discuss a number of experiments on motor competence acquisition.

4.1.5 Problems

The Perceptron, which I have used to store the associations between input stimuli (sensor signals) and motor actions of the robot has the disadvantage that it can only learn functions that are linearly separable. These are functions for which the input space can be separated into two clusters by a linear function[3]: into those input vectors that produce a "1", and into those that produce a "0" in the output unit. The logical *and* function is an example for such a learnable function. Here is an example of a function the robot could not learn using a Perceptron: suppose we wanted Alder to escape from a dead end by turning left whenever any of two front whiskers is on, and by reversing whenever both whiskers are on simultaneously. For the 'turn-left output node' of the Perceptron this means that it has to be on if either of the two whiskers fires, and to be off in the other two cases. This is the *exclusive or* function, a function that cannot be separated linearly and is therefore unlearnable by a Perceptron output node. Fortunately, the functions Alder and Cairngorm have to learn are linearly separable, as I will demonstrate shortly. This example, however, shows a property of Perceptrons that will often enable the robot to remain operational even if the correct function cannot be separated linearly: the output node learns a function that is similar to the correct function, and often it is close enough to the correct function to keep the robot in operation. In this case, the 'turn-left node' could learn to be on whenever at least one whisker is on (*or* function), this would make the robot leave the dead end as well.

[3] A hyperplane in the most general case.

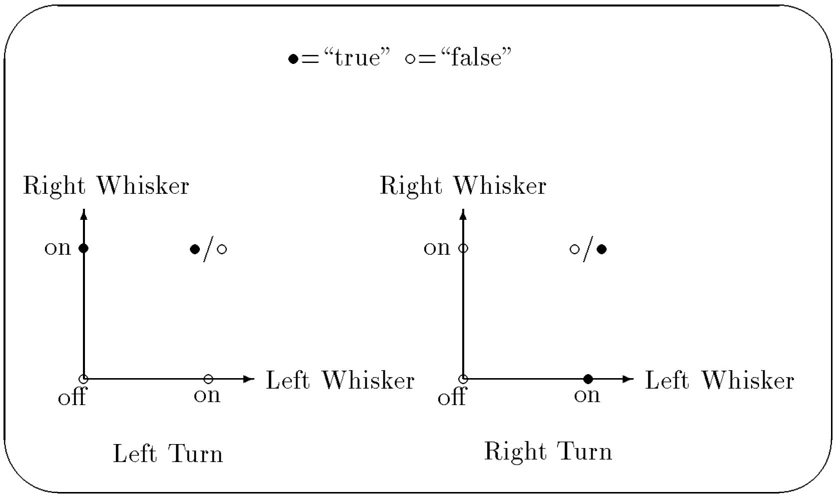

Figure 4.3: Learning to avoid obstacles is a linearly separable function.

4.1.6 Linear Separability

I will now take a closer look at the input vectors and the functions to be learned by the robot in the various experiments, to determine whether they can be separated linearly and therefore learned by a Perceptron.

Obstacle Avoidance

The simplest experiment is the one in obstacle avoidance. Using a two-bit input vector containing the status of the left and the right whisker (this vector is shown in figure 4.12) the robot has to turn away from a whisker that is on. That this function can be separated by a line is shown in figure 4.3 (this diagram refers to a robot with two tactile sensors, like the one shown in figure 4.8).

If the forward movement of the robot has to be learned as well, using the forward motion sensor, an additional two bits are put into the input vector: 'moving forward' and 'not moving forward' (this input vector is shown in figure 4.16). To accomplish this task, the 'forward node' of the Perceptron is on if the 'moving forward' bit is set, the 'reverse node' of the net is on if the 'not moving forward' bit is set. Again, this is a linearly separable function. Because of the way the controller is set up (the first instinct violated dominates the robot actions) one can ignore the signals from the whiskers when the robot is not moving forward. When the robot *is* moving forward, the additional 'moving forward' bit does not affect the linear separability of the obstacle avoidance function discussed earlier (just imagine a zero weight on all lines from that bit).

Wall Following

Alder and Cairngorm use a five-bit input vector to learn to follow walls. It contains the current two whisker readings, the previous two readings and a fifth bit that is only on if the other four bits are off[4]. This input vector is shown in figure 4.20. It is easy to show that the robots can find a satisfactory wall following behaviour by taking into account only the current two sensor readings and the fifth bit (bits three and four — the previous two whisker readings — can be used to escape from dead ends quicker, but for pure wall following they are not needed). A truth table for this simplified function is shown in figure 4.4. In this truth table it is assumed that the wall is on the left hand side of the robot. B1, B2 and B5 stand for the respective bits of the input vector shown in figure 4.20.

It is obvious from this truth table that the functions the turn-left node and the turn-right node have to learn are linearly separable: the turn-left node performs a $\overline{B1}$ function, the turn-right node a *B1* function. If the wall is on the right hand side of the robot, the respective opposite functions are performed.

[4]Instead of this fifth bit, a bias unit which is constantly on could also be used.

B1	B2	B5	Turn left	Turn right	
0	0	1	on	off	Boredom
0	1	0	on	off	Obstacle
1	0	0	off	on	Obstacle
1	1	0	off	on	Obstacle

Figure 4.4: Truth table for the wall following function.

Corridor Following

For corridor following, Alder and Cairngorm use the same five-bit input vector they use to learn wall following (figure 4.20). Again, it can be shown that it is possible to find a linearly separable function that results in a corridor following behaviour. To make the discussion easier, I will make the following assumptions: the robot has been running for a little while, which means that the input vector 00001 (no contact ever made so far) does not have to be considered. I also assume that the robot is not in a dead end, but between two walls, i.e. in a corridor. That means that the input vectors 11xx0 and xx110 (both whiskers on at time t or at time t-1) do not have to be considered. Figure 4.5 shows the Karnaugh diagram for the function to be performed by the 'turn-left node' of the Perceptron. B1 to B4 denote the first four bits of the input vector shown in figure 4.20.

This function is linearly separable, in figure 4.6 one can see that there exists a plane that separates the three input vectors that switch the turn-left behaviour on from the remaining five that switch it off.

Fast Learning

The advantage of the pattern associator that makes it superior to the backpropagation network (see appendix A) and reinforcement learning

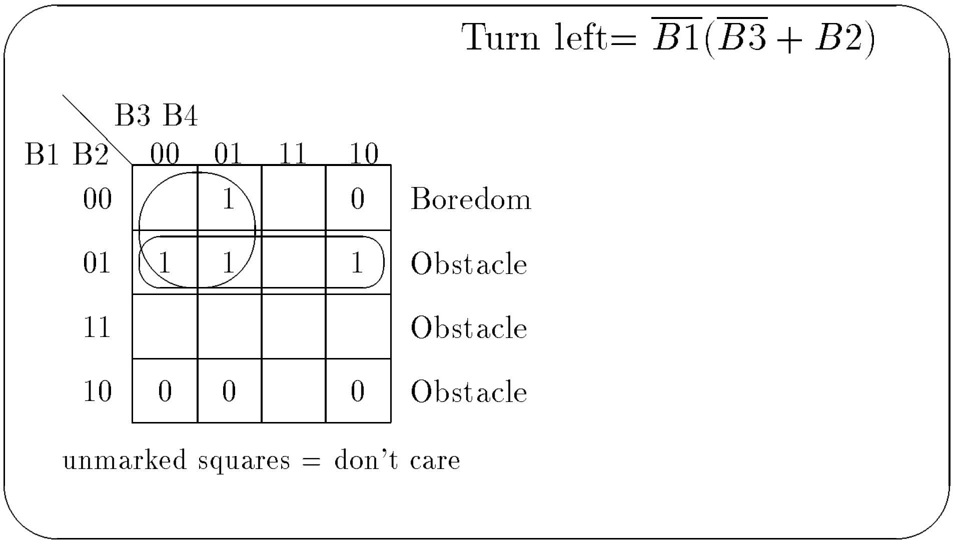

Figure 4.5: Karnaugh diagram for the function performed by the 'turn-left node' in corridor following.

is its extremely fast learning. A very small number of teaching experiences suffices to produce the correct associations between stimulus and response. A backpropagation network may typically require several hundred teaching experiences before a function is learned. Re-learning (for example when adjusting to new circumstances) then again takes typically several hundred training steps whereas the pattern associator re-learns as quickly as it learned in the first place. This property is important in robotics — certain competences, such as for example obstacle avoidance, have to be learned very quickly, because the robot's ability to stay operational crucially depends on them. In addition to this, [Brooks 91b] identifies as a further problem that the learning rate of the backpropagation network has to be set by hand in most cases. Consequently, most examples of robot controllers having a learning capability use techniques other than backpropagation networks ([Brooks 91b]). I cannot confirm the latter finding regarding

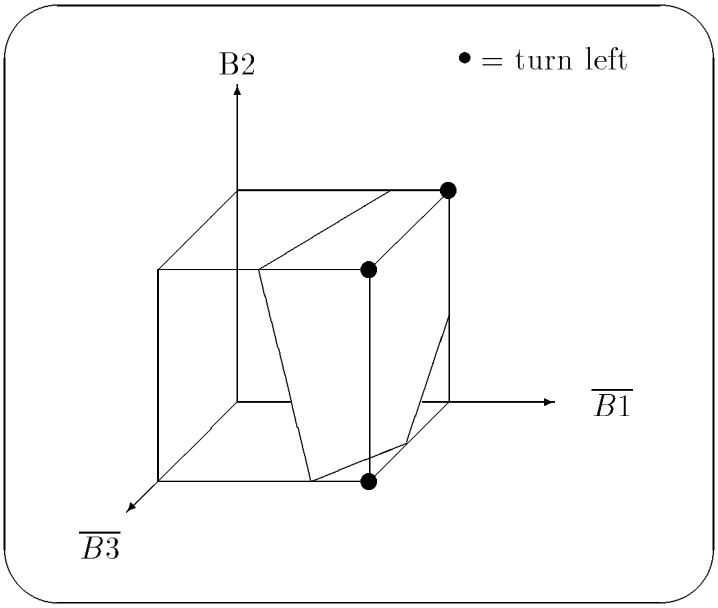

Figure 4.6: This 3D drawing shows that the function given in figure 4.5, which achieves corridor following, is linearly separable.

the manually set learning rate, as I have not used backpropagation networks here, but reducing the amount of *magic numbers*[5] is important for building adaptable and flexible robots, this problem has come up in the experiments in mapbuilding reported in chaper 5 of this thesis.

That the Perceptron is a fast learner is proven in the following section.

4.1.7 Convergence in the Perceptron

The following derivation of the convergence speed of the Perceptron is largely based upon [Beale & Jackson 90, pp.54ff], but has been aug-

[5]A magic number is a parameter that influences the success or failure of an algorithm and that is set by the experimenter, using his ingenuity to determine its value.

mented by the example shown in figure 4.7. Errors in Beale and Jackson's equivalent to the equations before and after 4.4 as well as equation 4.4 itself have been corrected.

Assuming that the input patterns to the Perceptron unit come from two classes, **F+** (generating a positive response) and **F-** (generating a negative response). **F** is the union of these two classes, i.e. the whole input space. The input is a vector called $\vec{x}$, this vector is assumed to be of length one to make the proof easier (this does not affect the general validity of the proof). The weight vector of the unit is $\vec{w}$. Adopting the convention that $\vec{x}$ is replaced by $-\vec{x}$ if $\vec{x}$ belongs to **F-**, then the Perceptron learning algorithm can be written as follows:

- Start: choose any $\vec{w}$.
- Test: choose any $\vec{x}$ from **F**.
  - If $\vec{w} \cdot \vec{x} > 0$ goto Test (this is the case where the Perceptron has classified correctly)
  - otherwise goto Update (this is the case where the Perceptron has classified wrongly).
- Update:

  $$\vec{w}_{t+1} = \vec{w}_t + \vec{x}. \tag{4.1}$$

  Goto Test.

The question looked at here is this: how often will the Perceptron learning algorithm go through the Update step until the error of the output unit lies below a particular threshold?

Consider figure 4.7, which shows a simple two-dimensional case.

All vectors $\vec{x}$ above the x-axis belong to class **F+**, all those below to class **F-**. The ideal weight vector of the Perceptron that would split these two classes is the one orthogonal to the dividing line between the

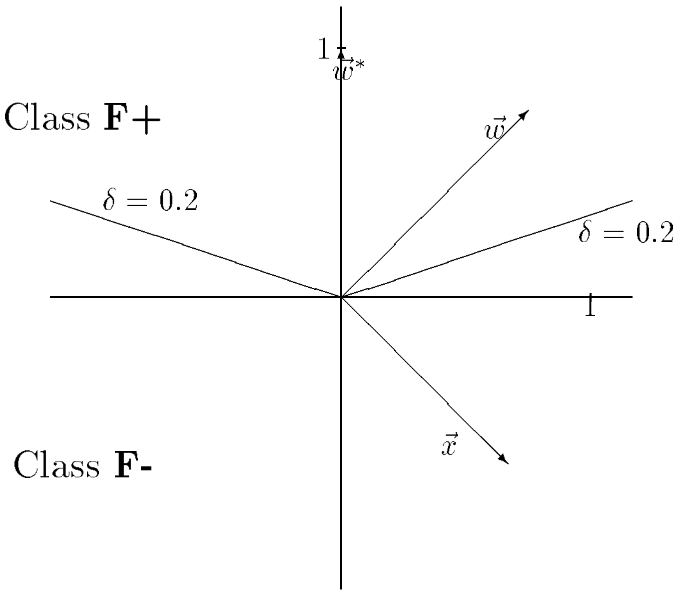

Figure 4.7:

classes (the x-axis in this case). I call this vector $\vec{w}^*$(again, to make the proof easier this is assumed to be a unit vector).

The input vectors $\vec{x}$ most difficult to learn are the ones very close to the x-axis, i.e. furthest away from the ideal weight vector $\vec{w}^*$. For these input vectors the scalar product $\vec{w}^* \cdot \vec{x}$ would be zero. We can define a small positive threshold δ which determines how close to the class dividing line (the x-axis) input vectors $\vec{x}$ may lie in order to be classified correctly:

$$\vec{w}^* \cdot \vec{x} > \delta. \tag{4.2}$$

The cosine between the actual weight vector $\vec{w}$ and the ideal weight vector $\vec{w}^*$ is

$$cos\phi = \frac{\vec{w} \cdot \vec{w}^*}{|\vec{w}||\vec{w}^*|}.$$

Because $\vec{w}^*$ is a unit vector, and because the cosine is never greater than one, this can be rewritten to

$$cos\phi = \frac{\vec{w} \cdot \vec{w}^*}{|\vec{w}|} \leq 1. \tag{4.3}$$

Every time the Perceptron learning rule runs through the Update routine, the angle between $\vec{w}$ and $\vec{w}^*$ changes, and so does the cosine of that angle. Because of equation 4.1, for one cyle through Update the numerator changes such:

$$\begin{aligned} \vec{w}^* \cdot \vec{w}_{t+1} &= \vec{w}^* \cdot (\vec{w}_t + \vec{x}) \\ &= \vec{w}^* \cdot \vec{w}_t + \vec{w}^* \cdot \vec{x} \end{aligned}$$

which yields the inequality

$$\vec{w}^* \cdot \vec{w}_{t+1} > \vec{w}^* \cdot \vec{w}_t + \delta$$

because $\vec{w}^* \cdot \vec{x} > \delta$ (see equation 4.2).

For n cycles through Update this becomes

$$\vec{w}^* \cdot \vec{w}_n > n\delta. \tag{4.4}$$

For one cycle through Update the denominator in equation 4.3 behaves as follows:

$$|\vec{w}_{t+1}|^2 = |(\vec{w}_t + \vec{x})(\vec{w}_t + \vec{x})| = |\vec{w}_t|^2 + 2\vec{w}_t \cdot \vec{x} + |\vec{x}|^2.$$

As $|\vec{x}|^2 = 1$ ($\vec{x}$ is a unit vector) and because $\vec{w}_t \cdot \vec{x}$ must be negative (otherwise the Update routine would not have been called), this can be rewritten as

$$|\vec{w}_{t+1}|^2 < |\vec{w}_t|^2 + 1,$$

for n cycles through Update the following inequality holds:

$$|\vec{w}_n|^2 < n. \tag{4.5}$$

Using equations 4.4 and 4.5, equation 4.3 can be rewritten as

$$cos\phi_n = \frac{\vec{w}_n \cdot \vec{w}^*}{|\vec{w}_n|}$$

$$cos\phi_n > \frac{n\delta}{\sqrt{n}}$$

Because of $cos\phi \leq 1$ (equation 4.3) this can be rewritten as

$$n \leq \frac{1}{\delta^2}. \tag{4.6}$$

This shows that the Perceptron will find the weight vector $\vec{w}^*$ that separates the two classes in a finite number n of cycles through Update, for $\delta = 0.2$ (shown in figure 4.7) at most 25 learning steps are required.

4.2 Outline of Competence Acquisition Experiments

The following sections of this chapter describe the various experiments conducted with Alder and Cairngorm, concerning the autonomous acquisition of motor competences.

All experiments described in section 4.3 concern the obstacle avoidance competence. Section 4.3.1 presents the experiment in which Alder and Cairngorm learn to avoid obstacles, using their whisker sensors. In the following experiment (section 4.3.3) Alder learns not only to use its whiskers to avoid obstacles, but also its forward motion sensor in order to learn which motor command will result in a forward movement. This means that the wiring of the motor power supply may be arbitrary, because the robot determines the correct motor command autonomously. The acquisition of an obstacle avoidance skill is again demonstrated in section 4.3.5, this time using the ultrasonic range finder of Alder.

Section 4.4 discusses an extension to the previous experiments. By adding another instinct-rule and enlarging the input vector to the network, the robots acquire a wall following skill; this is further extended in section 4.5 by adding yet another instinct-rule, which leads to a corridor following competence of both robots.

Finally, section 4.7 describes an experiment where Alder learns different mappings between sensors and actuators, depending on the context the robot is in (context sensitive learning).

4.3 Obstacle Avoidance

One of the most important tasks to achieve on a mobile robot is obstacle avoidance. When implemented on a robot, it becomes clear that even this seemingly simple task is not an easy one. Sensors give inconsistent readings, actuators depend on changeable factors such as supply current and voltage, wear, temperature, humidity and others. 'Avoiding Obstacles' is by no means a bygone research topic — it is still a widely discussed topic, see for example [Freund *et al.* 91] and [IEEE 91]).

Assume a robot had two whiskers mounted at the front of the vehicle and two independent motors that allowed the robot to move forward and backward as well as turn left or right. Such a robot is shown in figure 4.8.

If the task of this vehicle was to turn away from a touched whisker, that is to turn left if the right whisker is touched and vice versa, one could simply let the whiskers reverse the motors as shown in figure 4.9.

This would result in a vehicle avoiding obstacles. It could cope with new obstacles (it would, for example, move backwards if both whiskers were touched simultaneously, although the wiring may have been designed without this situation in mind), however it would not be able to cope with certain other changes. If, for example, its whiskers were swapped, the vehicle shown in figure 4.9 would move towards an

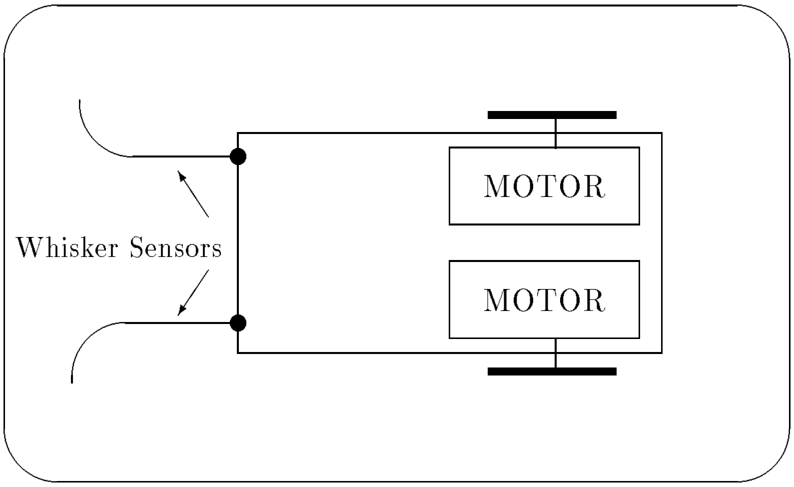

Figure 4.8: A simple vehicle.

obstacle instead of away from it!

As described earlier in this chapter, on the Really Useful Robots, I have replaced the coupling between sensor signals and motor actions by an artificial neural network. It is therefore possible for the robot to determine the wiring between sensors and actuators for itself. As a result of this Alder and Cairngorm are able to cope with unforeseen situations, such as swapped whiskers. This example may seem far fetched and unlikely to occur in reality, but the wiring of sensors can be error prone if a large number of sensors is to be connected. If the robot can determine the best wiring autonomously, malfunctions due to wrong sensor connections can be avoided. [Maes & Brooks 90]:

> "For more complicated robots prewiring solutions become either too difficult or impractical. ... Additionally, it is often too difficult for the programmer to fully grasp the peculiarities of the task and the environment, so as to be able to specify what will make the robot successfully achieve the task."

This is one of the experiments conducted: after having swapped the whiskers the robots are confused for a short time (typically they

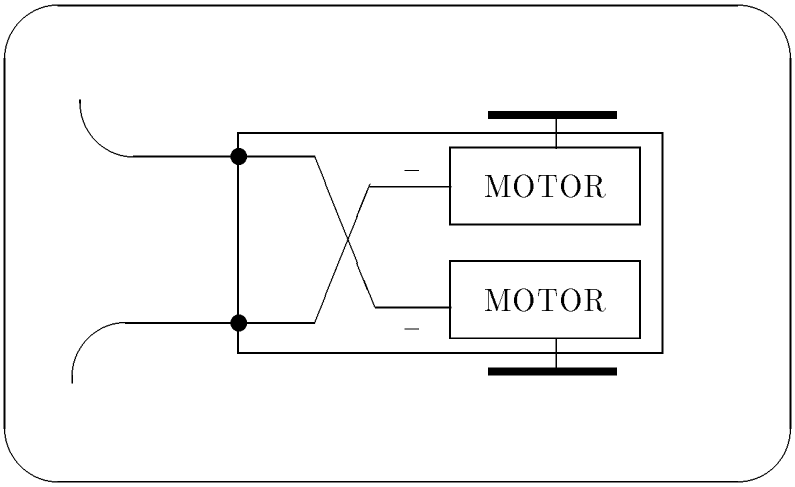

Figure 4.9: A simple obstacle avoiding vehicle.

make about two wrong decisions before they are able to avoid obstacles again), but then they re-train the network and acquire the obstacle avoidance competence again[6].

I performed two experiments on obstacle avoidance — in the first one Alder and Cairngorm learn to avoid obstacles, using only their whisker sensors, in the second one Alder uses its whiskers, as before, but also its forward motion sensor and therefore learns both to move forward and to avoid obstacles.

4.3.1 Obstacle Avoidance without using the Forward Motion Sensor

In the first experiment, which was conducted both with Alder and with Cairngorm, the robots were not able to detect forward motion. The

[6][Waterman 1989, p. 135] reports that North American desert ants *Novomersor* are able to follow a pheromone trail, albeit crookedly, if their antennae (their chemoreceptors) are swapped.

first instinct-rule, mentioned below, therefore could not be monitored[7]. The robots merely "assumed" that they were indeed moving forward when a 'move forward!' command was given. This meant that the wiring of the motors had to be correct (i.e., such that the robot really would move forward if a 'move forward' command was given).

The structure of the controller used for this experiment is shown in figure 4.10, the instinct-rules contained in the monitor are shown in figure 4.11.

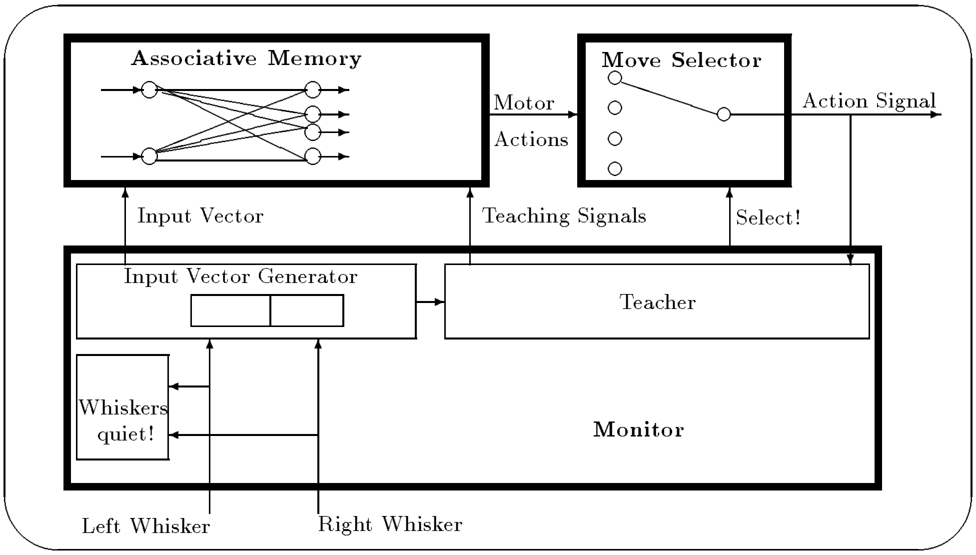

Figure 4.10: Controller used for obstacle avoidance I.

---

1. (Move forward!)
2. Keep whiskers 'quiet'!

Figure 4.11: Instinct-rules used for obstacle avoidance I.

[7]I have indicated this by writing this instinct-rule in brackets.

The network had only two input nodes, the input vector presented to the network is shown in figure 4.12.

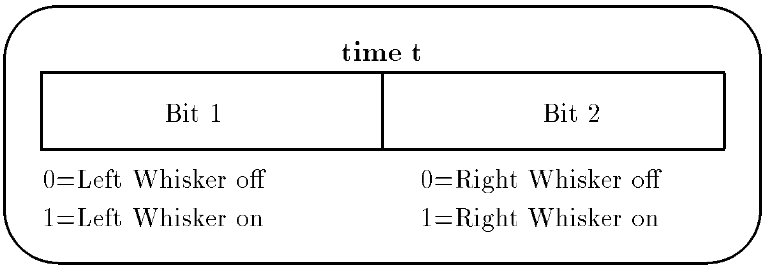

Figure 4.12: The input vector for obstacle avoidance behaviour I.

Both robots learned the effective wiring that would make them satisfy their instinct-rules very quickly. Within two to three learning steps the robots were able to turn away from obstacles. This takes less than a minute.

Using these two instinct-rules for assessing their behaviour, and the input vector shown in figure 4.12 the robots are not only able to acquire the obstacle avoidance competence and to re-learn it in the case of something drastic changing, like, for instance, the position of their whiskers; the robots are also able to cope with changed environments.

One example of such a changed environment (and task, whichever way you see it) is that of Alder and Cairngorm first learning to avoid obstacles, and then encountering a dead end. On page 49 I have described this experiment: although the robots enter the dead end with an obstacle avoidance behaviour (i.e., they turn in the opposite direction of a whisker that is touching something), they come out of the dead end turning in the same direction, regardless of which whisker is signalling. The effective wiring of the robots before and after the encounter of the dead end is shown in figure 4.13, I call the resulting behaviour *dead end escape behaviour*.

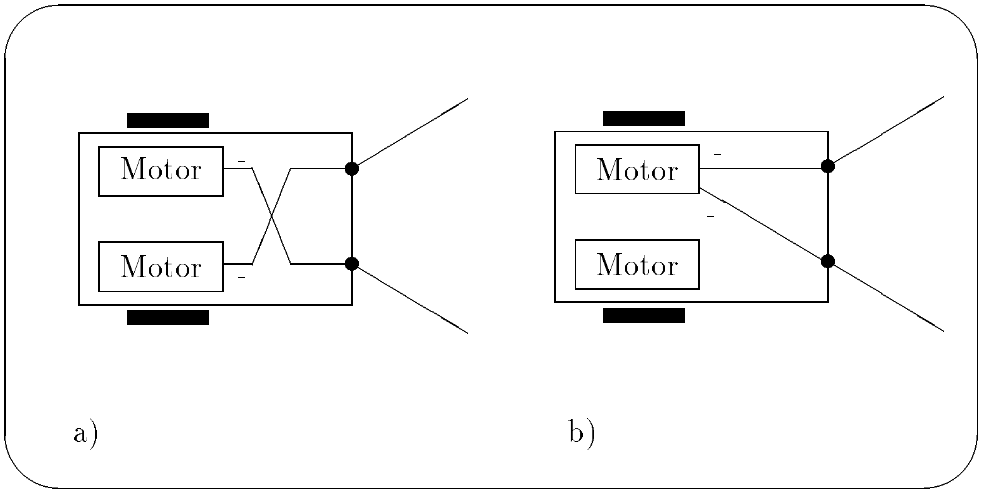

Figure 4.13: The effective wiring of Alder and Cairngorm before (a) and after (b) encountering a dead end.

A photograph of the obstacle avoidance experiment is shown after page 100. When Alder touched the wall with its right whisker for the first time (point A), it tried a right turn in order to move away from the wall (big line to the right in the photograph). This, of course, did not take the robot away from the wall; but the subsequent left turn did (curved line to the left). The next time the robot touched the wall with its right whisker (point B), it had learned this association and turned left (curved line to the left again). It maintained this behaviour throughout the whole experiment (points C and D).

4.3.2 Summary

The Experiment at a Glance
Title of Experiment: Acquisition of obstacle avoidance competence. **Robots used:** Alder and Cairngorm. **Sensors used:** Two tactile sensors, mounted at the front of the robots. **Motor actions used:** Left and right turn. (Move forward by default.) **Network used:** Two layer Perceptron with two input nodes and four output nodes. **Input vector used:** Two bit input vector containing the status of the two whiskers (see figure 4.12). **Instinct-rules used:** 1. (Move forward!) 2. Keep whiskers quiet!, see figure 4.11. **Magic numbers used:** The learning rate η of the Perceptron was chosen between 0.2 and 0.3, its value not being very critical. **Results and observations:** Alder and Cairngorm successfully acquire the obstacle avoidance competence. They can adapt to changing circumstances and regain the obstacle avoidance competence when unforeseen situations occur, for example when whiskers are swapped. When placed in a dead end the robots find the way out, often changing their behaviour from an obstacle avoiding one to a behaviour where they turn in the same direction, regardless of which whisker is signalling (dead end escape behaviour).

Following page: photograph showing the robot learning to avoid obstacles (explanation on page 97).

4.3.3 Learning to Avoid Obstacles and to Move Forward

In the second experiment Alder was equipped with a sensor to detect forward motion. Both instincts shown in figure 4.15 were therefore monitored — the controller for this experiment is shown in figure 4.14.

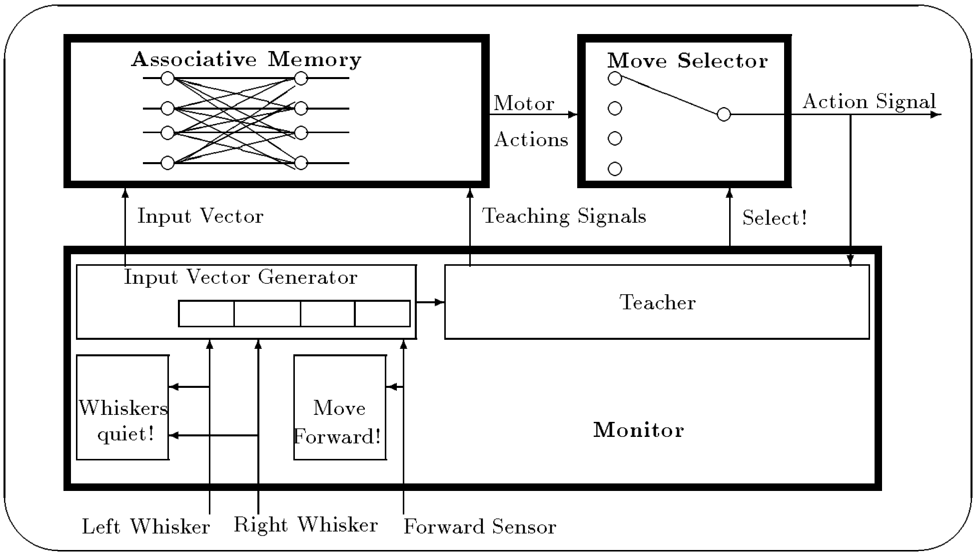

Figure 4.14: Controller used for Obstacle Avoidance II

---

1. Move forward!
2. Keep whiskers quiet!

Figure 4.15: Instinct-rules used for obstacle avoidance II.

As before, Alder was able to acquire the obstacle avoidance competence and cope with swapped whiskers. In addition to that the robot could now also cope with a reversal of the motor connections. In fact, it now didn't matter which way round the motors were connected, nor did it matter which way the sensors were connected. Because of the self-organising structure of the controller Alder was able to find the right behaviour to meet its instinct-rules. The input vector used for this second experiment is shown in figure 4.16. As the Pattern Associator can only process non-zero inputs, a two bit encoding of the forward motion sensor is used. Strictly speaking, the whisker sensor signals ought to be encoded in two bits as well, but because the no-whisker-touched state is not to be associated with any motor action other than forward (which the signal from the forward motion sensor takes care of) this is not necessary. If a Perceptron with input signals of ± 1 was used, the whole question would not arise, of course.

Left Whisker	Right Whisker	1=Moving forward 0=not moving forward	1=not moving forward 0=moving forward

Figure 4.16: The input vector used for obstacle avoidance behaviour II.

Again, as in the experiment mentioned in the previous section, Alder quickly learned the effective wiring between sensors and actuators. If the caster wheel of Alder is not in the 'forward' position when the robot is switched on, Alder will try the motor action associated with the first output node for a while. If in the course of this the forward motion sensor comes on, all instinct-rules are satisfied and the robot will move forward until one of the whiskers touches something. When this happens, Alder determines the effective obstacle avoidance behaviour as in the previous experiment. Note that in this experiment the motor actions associated with each of the four output nodes depend on the wiring of the motor power supply. For all other experiments described in this thesis the motors are wired up such that the nodes mean left turn, right turn, forward (or left turn) and backward (or right turn) in that order, but here deliberately the wires are connected arbitrarily, which means that the effect of each node has to be found out by the robot. To learn to move forward takes the robot about two to three learning steps (which take less than a minute), to learn obstacle avoidance takes about the same amount of time.

4.3.4 Summary

The Experiment at a Glance
Title of Experiment: Acquisition of obstacle avoidance and forward movement competences.
Robot used: Alder.
Sensors used: Two tactile sensors, mounted at the front of the robot, and one forward motion detector.
Motor actions used: Left and right turn, forward and reverse movement.
Network used: Two layer Perceptron with four input nodes and four output nodes(see figure 4.14).
Input vector used: Four bit input vector containing the status of the two whiskers and the status of the forward motion sensor, encoded in two bits (see figure 4.16).
Instinct-rules used: 1. Move forward! 2. Keep whiskers quiet!, see figure 4.15.
Magic numbers used: The learning rate η of the Perceptron was set to 0.2, its value was not critical.
Results and observations: Alder autonomously learns which motor command to issue in order to move forward, it also acquires an obstacle avoidance competence as in the previous experiment. Because forward motion is detected, the initial wiring of motor power supplies may be arbitrary, and may be changed during the experiment, too. As before, the robot is able to adapt to changing circumstances such as swapped whiskers or encountering a dead end.

4.3.5 Obstacle Avoidance using the Ultrasonic Range Finder

To show that the algorithm could be used for different types of sensors, the ultrasonic range finder of Alder was also used in experiments on acquiring obstacle avoidance behaviour. The controller used for this experiment was (almost) the same as the one shown in figure 4.10, but instead of using the signals from the left and the right whisker a three-bit signal constructed from range sensor data was used. A typical input vector is shown in figure 4.17.

1=range left $< \theta$	1=range middle$< \theta$	1=range right$< \theta$

Figure 4.17: The input vector used for obstacle avoidance behaviour, using the ultrasonic range sensor.

To construct this input vector, Alder performed turns to the left and to the right at regular intervals and used the ultrasonic range finder to determine the distance to the nearest object in these directions, as well as straight ahead. The θ shown in figure 4.17 is a threshold, it is typically set at about 30 cm.

The instinct-rules used in this experiment are similar to the instinct rules used in previous experiments. Instead of the Keep Whiskers Quiet! rule, however, the signals from the ultrasonic range finder trigger the second instinct-rule, as shown in figure 4.18.

As in previous experiments, effective associations between the input vector and motor actions evolved, Alder learned successfully to avoid obstacles. Unlike in the previous experiments, where Alder touched

1. (Move forward!)

2. Keep ultrasonic readings below threshold!

Figure 4.18: Instinct-rules used for obstacle avoidance with the ultrasonic sensor.

obstacles before it was able to move away from them, here the robot never touched the obstacles. Even though ultrasonic sensors give varying readings, dependent on the surface structure of the obstacle and the angle under which it is perceived, the robot always turned away from them before making contact. Due to the nature of the sensor used the turning movement was not always initiated when the threshold distance was reached, but always before the robot had arrived at the obstacle.

As before, learning was fast. Within two or three learning steps the robot learned the effective wiring between the input vector derived from the ultrasonic sensor, and the motor actions.

4.3.6 Summary

The Experiment at a Glance
Title of Experiment: Acquisition of obstacle avoidance competence.
Robots used: Alder.
Sensor used: One ultrasonic range finder, mounted at the front of the robot. Active range 20cm to 3m.
Motor actions used: Left and right turn. (Move forward by default.)
Network used: Two layer Perceptron with three input nodes and four output nodes.
Input vector used: Three bit input vector, stating whether the ultrasonic range reading to the left, the middle and the right of the robot was below a preset threshold. (see figure 4.17).
Instinct-rules used: 1. (Move forward!) 2. Keep ultrasonic readings below threshold! — see figure 4.18.
Magic numbers used: The learning rate η=0.2, threshold θ for range$\approx$45cm. Both values are not critical.
Results and observations: Alder successfully acquires an obstacle avoidance competence without actually touching obstacles, using the ultrasonic range finder.

4.4 Wall Following

A further advantage of a self-organising controller as shown in figure 4.1 is that the behavioural repertoire of the robot can quite easily be expanded by adding further instinct-rules (sometimes the input vector to the associative memory also needs to be changed).

The second experiment I conducted showed this: the robot's task still was to move away from obstacles, but in addition to that the robot now also had to stay close to a wall and follow it. This was achieved by using one additional instinct-rule, "boredom", so that the whole set of instincts looked like figure 4.19.

---

1. (Move forward!)
2. Keep whiskers quiet!
3. Make whiskers signal after a while!

Figure 4.19: Instinct-rules used for wall following

The input vector used now contained some history, it is shown in figure 4.20, the controller for this experiment is shown in figure 4.21.

Using this set of instinct-rules and this input vector, the robots were able to learn to follow a wall successfully. When the robot is started, having been placed near a wall, but not touching it, the internal clock that triggers instinct-rule number 3 (see figure 4.19) is reset. This means that neither instinct-rule number 2 nor number 3 trigger, but number 1 does: the robot starts moving forward. After about four seconds the clock triggers rule number 3, and a wall seeking movement is initiated. Eventually the robot will touch the wall, thus satisfying rule 3 (and taking it off the "agenda"). Immediately, however, rule 2

Time t		Time t-1		
1=Lft. Wh. on 0=Lft. Wh. off	1=Rgt.Wh. on 0=Rgt.Wh. off	1=Lft. Wh. on 0=Lft. Wh. off	1=Rgt.Wh. on 0=Rgt.Wh. off	$\neg(B1 \vee B2 \vee \ldots$ $\ldots \vee B3 \vee B4)$
B1	B2	B3	B4	B5

Figure 4.20: Input vector used for wall following and corridor following.

will be violated, and the robot will start an obstacle avoidance action. Every time an instinct-rule is violated, an input vector as shown in figure 4.20 is generated and a motor action sought that will satisfy this particular instinct-rule. It does not matter if in the course of trying to satisfy this one instinct-rule other rules are violated, too. The robot attends to the first violated instinct-rule until it is met again, and then continues.

The experimental result of this was that after very few negative teaching signals Alder and Cairngorm learned which way to turn in order to touch the wall, and which way to turn in order to get away from it. As in the previous experiment in obstacle avoidance, the robots learn the correct moves after about two teaching steps. The whole wall following competence is acquired in under one minute. The path the robots describe is shown in a photograph after page 112. Initially, they tried to find the wall on the left hand side (big turn on the left in the photograph at point A), but found it to be on the right. To move away from the wall the robot used a left turn, which was successful. The photograph then shows small turns to the right to touch the wall (short peaks to the right), and left turns to move away from it (B).

Because of their ability to acquire the wall following competence,

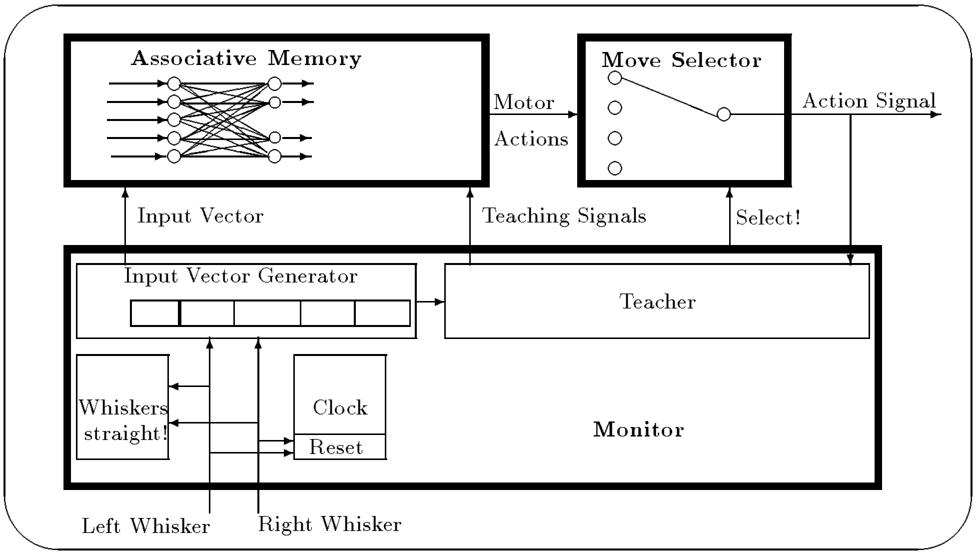

Figure 4.21: Controller used for wall following and corridor following.

the robots are not dependent on any particular wiring or experimental setup. The way the whiskers are connected to the interface and the side on which the wall is can be chosen arbitrarily, the robots will nevertheless learn how to follow the wall. The wall following experiment shows another advantage. Because of the structure of the controller, without any especially given knowledge, the robot can follow walls on either its left or its right hand side. This makes the robot more flexible, and better able to cope with situations such as corridors, where the wall on one side suddenly stops (because of a door, for example), but continues on the other side. At such places the robot would be temporarily "confused", but then learn to follow the wall on the opposite side and continue its path[8]. Also, the experiment shows that the behavioural repertoire of the robots can easily be expanded by adding instinct-rules, an observation that will be amplified by the following experiment.

[8]This, of course, only if the opposite wall was close enough to be reached by a whisker!

4.4.1 Summary

The Experiment at a Glance
Title of Experiment: Acquisition of wall following competence.
Robots used: Alder and Cairngorm.
Sensors used: Two tactile sensors, mounted at the front of the robots, and the internal clock.
Motor actions used: Left and right turn. (Move forward by default.)
Network used: Two layer Perceptron with five input nodes and four output nodes (see figure 4.21).
Input vector used: Five bit input vector, containing the current signals from the two whiskers and the whisker signals of the previous time step. Bit five is on only if bits one to four are all off (*negated or* function of the first four input bits) — see figure 4.20.
Instinct-rules used: 1. (Move forward!) 2. Keep whiskers quiet! 3. Make whiskers signal after a while! — see figure 4.19.
Magic numbers used: Learning rate η=0.2 to 0.3, wall seeking movement sets in approximately every 4 seconds. Both values are not critical.
Results and observations: The robots successfully learn which way to turn in order to touch a wall, and which way to turn in order to move away from it. If whiskers are swapped, walls are swapped or the direction of travel is changed, the robots adapt to the new situation and regain the wall following competence.

Following page: photograph showing the robot learning to follow a wall (explanation on page 109).

4.5 Corridor Following

As just stated in section 4.4, it is easy to extend the behavioural repertoire of the robot by adding further instinct-rules[9]. In order to obtain a corridor following behaviour, literally the only thing I had to do was to add a fourth instinct-rule (about five lines of program code) — the input vector used was the same as the one used for wall following, shown in figure 4.20, so was the controller architecture. The set of instincts to acquire the corridor following competence are shown in figure 4.22.

---

1. (Move forward!)
2. Keep whiskers quiet!
3. Make whiskers signal after a while!
4. Make alternate whiskers signal!

Figure 4.22: Instinct-rules used for corridor following.

The robot now learns which way to turn to touch alternate walls of a corridor, and which way to turn to move away from either wall. As before, learning is fast, the robots acquire the corridor following competence in less than a minute. This last experiment demonstrates how easy it can be to "program" the robot to do new tasks, using the self-organising controller described here.

[9] That is, once the required instinct-rule is known, it is easy. See also the open questions raised in section 7.2.1.

4.5.1 Summary

The Experiment at a Glance
Title of Experiment: Acquisition of corridor following competence.
Robots used: Alder and Cairngorm.
Sensors used: Two tactile sensors, mounted at the front of the robots, and the internal clock.
Motor actions used: Left and right turn. (Move forward by default.)
Network used: Two layer Perceptron with five input nodes and four output nodes (see figure 4.21).
Input vector used: Five bit input vector, containing the current signals from the two whiskers and the whisker signals of the previous time step. Bit five is on only if bits one to four are all off (*negated or* function of the first four input bits) — see figure 4.20.
Instinct-rules used: 1. (Move forward!) 2. Keep whiskers quiet! 3. Make whiskers signal after a while! 4. Make alternate whiskers signal! — see figure 4.22.
Magic numbers used: Learning rate η=0.2 to 0.3, wall seeking movement sets in approximately every 4 seconds. Both values are not critical.
Results and observations: Alder and Cairngorm learn which way to turn in order to touch left and right wall of a corridor in turns, and which way to turn in order to move away from them again. This experiment shows how easy it can be to increase the robot's behavioural repertoire, using this controller architecture.

4.6 Arbitration of Instinct Rules

It is not always immediately possible to extend the robot's behavioural repertoire by adding further instinct-rules. From obstacle avoidance to wall following the instinct-rule "Make whiskers signal after a while!" was added; to achieve corridor following, coming from wall following, the instinct-rule "Make alternate whiskers signal!" was added. This worked, because none of the instinct-rules contained in the set conflicted with any other instinct-rule. Only one instinct-rule at a time could be violated, and only that instinct-rule could be satisfied again.

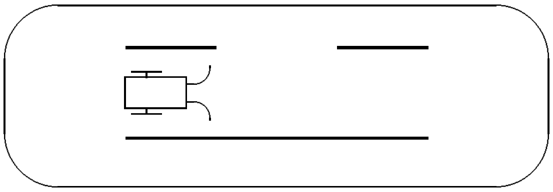

Figure 4.23: Wall and corridor following.

If, however, the task of the robot was to, for example, follow a wall or a corridor, as shown in figure 4.23, then neither the set of instinct-rules shown in figure 4.19 nor the one shown in figure 4.22 would make the robot learn the correct behaviour, because the instinct-rule "Make alternate whiskers signal!" cannot be satisfied in a non-corridor part of the robot's world.

Possible solutions to this problem include

1. Use only non-conflicting instincts, or
2. add further instincts for arbitration, or
3. define some other arbitration strategy.

In the case of the example just mentioned — a robot follow a corridor which does not have continous walls on both sides — the following set of instinct-rules could be tried:

1. Move forward!
2. Keep whiskers quiet!
3. Make whiskers signal after a while!
4. For the first n seconds try to make alternate whiskers signal, after that period any whisker!

I have not tried this set of instinct-rules in an actual experiment, but here is some speculation as to what might happen: rules 1 to 3 will result in a wall following behaviour, as described before. Rule 4 is identical to rule 4 in figure 4.22, as long as the robot is in a corridor. Therefore, in a corridor the robot should follow it, as described before in section 4.5, touching alternate walls. As soon as this fails, because of one wall ending, the robot will not be able to turn in such a way that the required whisker signals, at least not immediately. If the waiting period n is large, the robot might turn by 180 degrees and thus satisfy instinct-rule 4 (that would make the robot turn back whenever there is a gap in either of the two walls of a corridor). If the waiting period n is too short to allow the robot to turn back, rule 4 will become ineffective, resulting in a normal wall following behaviour being switched on and the robot should continue its path, following a wall. As long as the gap in one of the walls remains, this process should continue, until two walls are present again. Then the robot should switch back to a behaviour where it touches alternate walls.

4.7 Context-dependent Learning

On page 93 I described an experiment in which the robot had first learned to avoid obstacles, and then encountered a dead end. The obstacle avoidance behaviour is badly suited for leaving dead ends, because inevitably there will be the situation where the robot turns back into the dead end instead of turning away from it. Alder and Cairngorm discover that through the interaction with their environment and change their behaviour: after having encountered a dead end they turn in the same direction regardless of which whisker is touched. This is what I called dead end escape behaviour.

In the process of acquiring the dead end escape behaviour the previously acquired knowledge about obstacle avoidance is replaced by knowledge about dead end escape. Consequently, when Alder and Cairngorm leave a dead end and encounter an obstacle, they have to re-learn obstacle avoidance behaviour. They lack a notion of context. In this section I describe an experiment done on context dependent learning in which Cairngorm learns to distinguish two different contexts and behave appropriately in both.

The experimental setup is as shown in figure 4.24. The robot was placed in an enclosure which consisted of two dead ends and a corridor in between. The robot's task was to follow the corridor and to get out of the cul-de-sacs, resuming the corridor following behaviour.

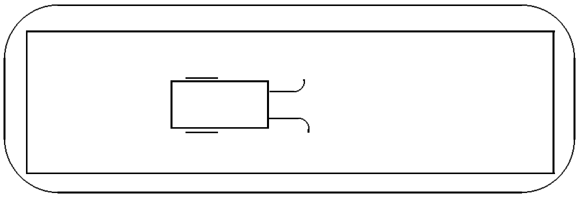

Figure 4.24: The experimental setup for context-dependent learning.

To find out whether contextual information could be exploited by the robot I conducted the following experiment. Cairngorm was controlled by the same controller that was previously used to acquire the corridor following competence, as described in section 4.5. The only difference to the controller described there was an additional bit in the input vector to the Pattern Associator (see figure 4.25).

Time t		Time t-1		
1=Left Wh. on 0=Left Wh. off	1=Right Wh. on 0=Right Wh. off	1=Left Wh. on 0=Left Wh. off	1=Right Wh. on 0=Right Wh. off	Switch

Figure 4.25: Input vector for context–dependent learning.

This last bit in the input vector was set whenever a switch on the robot was on, and it was reset whenever the switch was off. As this switch was manually operated, the operator could identify different contexts and make them available to the learning algorithm.

4.7.1 Experimental Results

The robot was placed in the enclosure shown in figure 4.24. Whenever the robot was in one of the two dead ends, that is in a dead end context, the switch was on. Whenever the robot was in between dead ends, that is in a corridor following context, the switch was off. After the robot had learned corridor following (and, in the process, encountered dead ends several times), the effective wiring of Cairngorm was analysed. The results are shown in figure 4.26: whenever the switch is off (corridor following context), the robot turns left when the right whisker is on, and right when the left whisker is on (figure 4.26a). When the switch is on (dead end escape context) the behaviour is reversed: the robot turns towards a touched whisker (figure 4.26b). Both these behaviours are appropriate for their respective contexts. When following a corridor, whiskers will only be on after a wall seeking movement. A turn away from a touched whisker will take the robot back into the middle of the corridor. In the dead end context this is different, here it is a turn towards a touched whisker that will take the robot to the exit fastest.

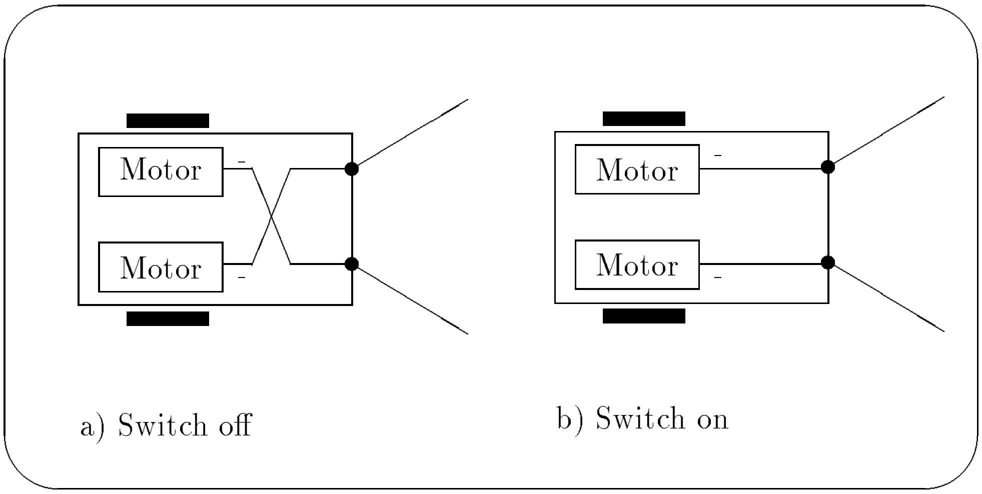

Figure 4.26: Results of context–dependent learning experiment.

The purpose of this experiment was to investigate whether the controller could successfully exploit contextual information. This is why the robot was provided with contextual information. A whole series of experiments is conceivable in which this contextual information is determined by the robot itself, not by the operator. Here are some examples: when the robot follows a corridor, it is highly unlikely that the same whisker will signal twice consecutively, because the robot turns left, say, but then turns back right and when initiating the next wall seeking move will turn right, therefore the last touch will have been on the left, the current one on the right. The only situation where the same whisker will signal more than once consecutively is if the robot is in a dead end: the fact that one whisker has signalled twice in a row can therefore be used as a detector for a dead end context. Similarly, if the robot had three front whiskers, one mounted at the left, one in the middle and one at the right, then the middle whisker would normally only be on in a dead end, not in a corridor. Instead of a middle whisker, a central sonar sensor would serve the same purpose.

4.7.2 Summary

The Experiment at a Glance
Title of Experiment: Exploiting contextual information in motor competence acquisition.
Robot used: Cairngorm.
Sensors used: Two tactile sensors, mounted at the front of the robot, and a manually operated switch.
Motor actions used: Left and right turn. (Move forward by default.)
Network used: Two layer Perceptron with five input nodes and four output nodes (see figure 4.21).
Input vector used: Five bit input vector, containing the current signals from the two whiskers and the whisker signals of the previous time step. Bit five is connected to a manually operated switch and is on if the switch is on — see figure 4.25.
Instinct-rules used: 1. (Move forward!) 2. Keep whiskers quiet! 3. Make whiskers signal after a while! 4. Make alternate whiskers signal!, see figure 4.22.
Magic numbers used: Learning rate η=0.3 (not critical to set).
Results and observations: In this experiment contextual information is supplied by means of the switch: it is off in a corridor following context, and on in a dead end context. The effective connections between sensors and actuators that emerge from this setup are such that the robot turns to the opposite side of a signalling whisker when the switch is off (corridor following), but turns towards the direction of a signalling whisker when the switch is on (dead end), see figure 4.26.

4.8 Discussion

Three major points contribute towards the fact that this scheme works:

1. Limited input space, possibly using quantised input data,
2. limited output space, and
3. a robust action selection scheme.

Limited input and output spaces means that the search space is small, it means that an effective motor action can quickly be found. The larger the search space, the greater the chance that the Perceptron will learn a suboptimal behaviour, and the greater also the computational cost.

Lastly, as I have already shown earlier on page 81, the action selection scheme is robust and finds effective motor actions even if the initial execution of the 'correct' motor actions was unsuccessful. This ensures that even in situations where the variation inherent in the world has made a particular action fail despite the fact that it should have succeeded, this action will nevertheless be found later on.

The Capacity of the Associative Memory

A Perceptron can reliably store 2N random input-output pairs, with N being the number of input units of the network ([Hertz *et al.* 91, p.111]).

This means that for every network with three or more input units its capacity (2N) is less than the number of possible input-vectors (2^N), i.e. that it is impossible to store an output pattern for *every* possible input signal. The ratio $\frac{2N}{2^N}$ becomes worse with increasing N, which means that the likelihood of the network's capacity being exhausted before all required patterns are stored increases with increasing N. However, this is not a problem in the experiments reported here, the network's capacity is sufficient for all experiments conducted with Alder and Cairngorm.

How to devise Instinct-Rules

Having accomplished wall following, it was extremely easy to increase the behavioural repertoire of the robot to corridor following by simply adding another instinct-rule, a mere five lines of code. The question is: how does one find the required set of instinct-rules? There is no algorithm yet for doing this, but here are some guidelines:

1. Instinct-rules are always associated with a sensor which monitors or measures some specified condition. Sensors in this sense can be:
   - Physical sensors (e.g., whiskers),
   - internal sensors (e.g., a clock), and
   - memory (e.g., "Make alternate whiskers signal!").
2. Instinct-rules are not behaviours. "Move towards the light!", therefore, is not an instinct-rule, but "Increase the reading from a light sensor!" is.
3. Instinct-rules generate the desired behaviours, the question therefore must be "Which sensor signal can be associated with the desired behaviour?".

4.9 Summary

In order to stay operational, mobile robots need the ability to move away from obstacles, both convex and concave. For higher level tasks such as navigation or delivery tasks, robots also need the ability to follow defined paths, as for example by following walls or following corridors. In order to achieve flexibility in unforeseen situations it is desirable if the robots can acquire these competences autonomously, as such an ability will allow them to adapt if circumstances change.

In a number of experiments in motor competence acquisition, Alder and Cairngorm learn how to move forward (thus determining the effective wiring required), how to avoid obstacles, how to follow walls and how to follow corridors. Using a self-organising controller whose central component is an associative memory that associates input stimuli with task-achieving motor actions, the robots are not only able to acquire the required motor competences in the first place, they are also able to adapt to situations such as reversed motor power supplies, changed sensor wiring and changed environments (such as new obstacle types or changed wall layout whilst following a wall).

The next chapter adresses one of the higher level tasks mentioned above, that of navigation. In order to navigate a robot needs to know a) where it is at the moment (location recognition) and b) how to get to the desired location (map interpretation). How location recognition can be achieved autonomously, again using connectionist computing architectures, is the topic I will now turn to.

Chapter 5

Location Recognition, using Self-Organising Feature Maps

5.1 Introduction

In the previous chapter I presented experiments on competence acquisition. The competences were achieved by establishing appropriate couplings between sensor signals and motor actions. This chapter addresses a different problem: can self-organising structures be used to acquire simple navigational skills? The task for Alder and Cairngorm here was to recognise locations in an enclosure (see figure 5.1) after having had some time to explore the enclosure.

When I started, Alder was programmed to move around in arbitrary directions, avoid obstacles it encountered and feed the sensor signals into a self-organising network (as described in section 2.4.2). The idea was that distinctive excitation patterns would develop on the network, a different one for each physical location. But nothing of that sort happened: instead, the excitation patterns seemed to be arbitrary in

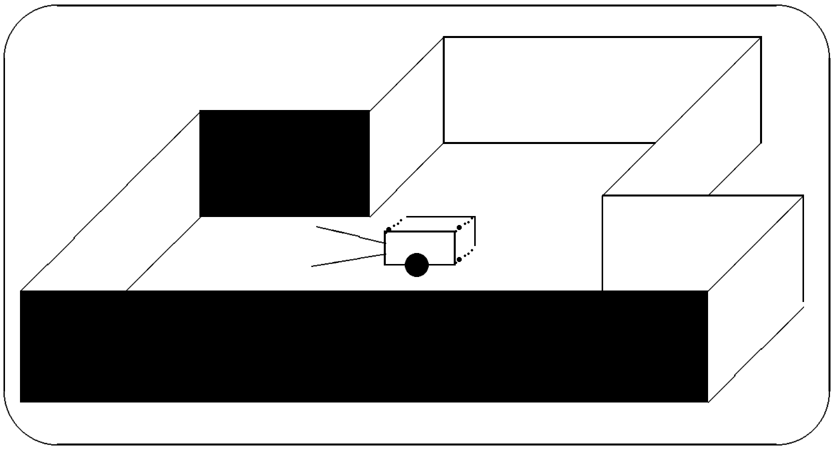

Figure 5.1: The experimental enclosure for location recognition.

regard to physical locations.

It was in a mountain hut (a "bothy") in February 1990 that I realised what had to be changed in order to make the scheme work. I had fought my way through a blizzard and finally arrived at the bothy, around midnight. It was pitch black dark, and my torch had broken. I had no idea what the hut was like, whether there was any furniture in there, how big it was. I started moving along the wall, groping in the dark. I followed the wall around the bothy and discovered corners, the fireplace and a pile of stones, a chest with mountain rescue equipment and the window sill (I also discovered that the hut was full of snow...). Sensing and acting are tightly coupled — only through appropriate action can meaningful sensor signals be derived and used for navigation or location recognition. Following the wall, rather than finding my way straight across the room had two advantages: it defined a canonical path, and along that path much more information was to be gathered than by going across the room (a rich sensory stream, rather than a thin sensory stream). Also, when not wall following it is difficult to define identical paths, a problem that is particularly relevant if location recognition schemes are to be implemented on mobile robots which are subject to noise and variation. I would not have got a good idea of the shape of the bothy by going straight across the room and turning as soon as I touched a wall. The same applies to the robots. After this weekend I changed the behaviour of Alder and Cairngorm to a wall following one for training a self-organising network. The scheme worked: unique excitation patterns developed for individual locations.

In this chapter I describe three experiments on location recognition. For all of these experiments the basic setup was the same: while the robot follows the perimeter of its enclosure, input stimuli are presented to a self-organising feature map (SOFM). The developing excitation patterns are then used to recognise locations in the world (identical excitation pattern = identical location).

The first experiment, described in section 5.2, used information about landmarks to train the self-organising net. As the robot explored its enclosure, it detected convex and concave corners, using a hand-programmed corner detector. Whenever such a landmark was found, an input vector containing information about current and previous landmarks was generated and used to train the net. This worked satisfactorily: the robot was able to recognise places.

Information about landmarks is specialised information (as opposed to general information). In the second experiment I therefore reduced the specificity of the information used to train the net. Instead of detecting landmarks, the robot merely registered all motor actions ever performed, and used this to shape the self-organising feature map. Most of the motor actions performed by Alder and Cairngorm have nothing to do with 'interesting' places such as corners. They have to do with the process of following a wall — turning right to find the wall, turning back left to realign with it — and therefore they carry very little information that is useful for recognising places. In order to succeed, the robots used seven independent self-organising feature maps, each one of these looking at a different length of history of motor actions. Again, Alder and Cairngorm were able to recognise locations. The experiment is described in more detail in section 5.3.

The third experiment is a link between the first two. On the one hand it uses just one network and is fairly easy to implement — like the first experiment — on the other hand it does not use landmarks, but motor actions — like the second experiment. Using *all* motor actions as input to the self-organising network and letting the robot itself make the distinction between significant and insignificant events is the most general approach that can be taken, and it follows the RUR philosophy of avoiding as much predefined knowledge as possible closely. However, it is computationally expensive to 'get rid of uninteresting motor actions', and a large number of externally set parameters are needed, too (more about this, as mentioned, in section 5.3). Therefore, in the third experiment, Alder and Cairngorm take only those motor actions into account that are significantly longer than the average — these motor actions usually happen at 'interesting locations'[1]. The third experiment is described in section 5.4.

5.2 Location Recognition using Sensor Signals

Alder's and Cairngorm's task was to recognise locations in their enclosure after they had explored it for a while. To explore, the robots followed the walls of the enclosure, completing a couple of circuits. 'Exploring' means that they followed the wall of the enclosure, detect-

[1] "Significantly longer" here means longer by a certain proportion which is specified by the designer.

ing convex and concave corners as they went along (detecting corners is not difficult: whenever the robot needs more than a certain, preset time to find the wall on its right, it has arrived at a convex corner. Similarly, if the robot needs more time than usual to get away from an obstacle, it has arrived at a concave corner).

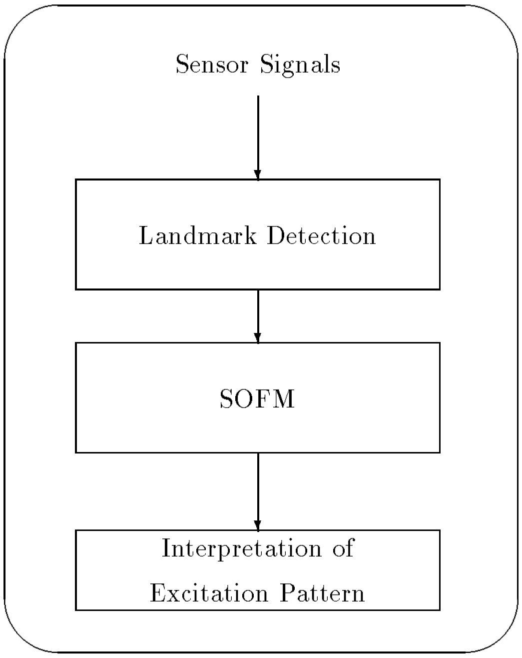

Figure 5.2: The method used for location recognition.

The method of location recognition that I used is depicted in figure 5.2 and it works like this: whenever a convex or concave corner was detected, an input vector was created and presented to the self-organising feature map. This input vector contained information about the type of corner the robot had detected, as well as information about the type of the previous corners. The simplest vector used is shown in figure 5.4[2]. The self-organising feature map that was used is shown in figure 5.3. It was a ring-shaped, one-dimensional network of fifty cells.

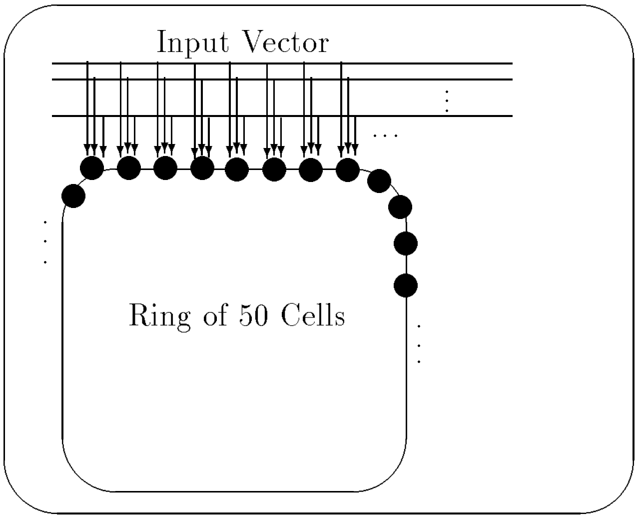

Figure 5.3: The self-organising network used on Alder.

Assuming that features are topologically isolated, the minimum number of network cells required can be determined as follows: if cells are updated within a neighbourhood of two cells around the most excited cell then five cells are updated each time and could theoretically represent one particular area in the robot's enclosure. As there are eight corners in Alder's enclosure, the minimum number of cells required is fourty cells; however in practice a larger number is needed because areas overlap. Being one-dimensional, the computational requirements are very low because each cell has only two neighbours, instead of eight[3] in a two-dimensional net. This is preferable for implementing the scheme on a robot with limited computing power, like

[2]Note that it is necessary to encode the information about convex and concave corners using 2 bits, because artificial neural networks can only make use of non-zero input lines (see also equation 2.5). Units giving non-zero outputs of ± 1 could also be used.

[3]See [Kohonen 88, p.132, Fig. 5.11a].

Alder. The shape of a ring helps avoid border effects. The behaviour of this network is as previously described in section 2.4.2, the neighbourhood within which weight vectors are updated is $\pm$ 2 cells (constant over time). This size of neighbourhood was chosen as a compromise between too large a neighbourhood, which means a high computational load, and too small a neighbourhood, which means that the topological mapping of the self-organising feature map will not have any effect.

The longer the robot went around in the enclosure, the more settled the ring became and the more precise the response to a particular input stimulus. After about three rounds a particular corner was 'marked' by pressing a pushbutton on the robot. The excitation pattern $\vec{\kappa_m}$ that the ring showed when the button was pressed was then stored. In order to detect when the robot had arrived back at the marked location all subsequent excitation patterns $\vec{\kappa}$ were compared with $\vec{\kappa_m}$ by calculating the Euclidean distance $||\vec{\kappa} - \vec{\kappa_m}||$ between them. Obviously, if the robot is able to construct a meaningful internal representation of its environment in the SOFM this difference should be small when the robot is at the marked corner, and it should be noticeably larger at any other corner. Here, the response of the whole network is used to recognise a location. Alternatively, the property that self-organising feature maps produce topology preserving mappings could be exploited: merely the coordinates of the excitation centres could be compared.

In summary, the location recognition mechanism looked like this:

1. Initialise the self-organising network (i.e. the ring) by filling the weight vectors of all cells with randomly chosen, evenly distributed values.

2. Normalise all the weight vectors.

3. Present an input stimulus to the ring.

4. Determine the response to this stimulus for each cell of the ring according to equation 2.4.

5. Determine the unit that is responding most strongly.

6. Update the weight vectors of the five units within a neighbourhood of $\pm$ 2 cells of the maximally responding cell, according to equation 2.5 given in chapter 2.

7. Normalise those five weight vectors again.

8. Continue with step 3.

5.2.1 Experimental Results

Whenever Alder or Cairngorm detected a corner, they generated an input vector and used it to train the self-organising feature map. The first input vector I used is shown in figure 5.4.

This vector contains information about the type of corner the robot is currently at, as well as information about the type of the previous corner. This is encoded in four binary values. The fifth component of the input vector is a real number, giving the number of completed wheel revolutions between the two corners, divided by twelve. Twelve, because the longest distance in the particular enclosure used (the distance G-H in figure 5.5) generates about fifteen revolutions, the shortest, one or two. By dividing this count by twelve the last component of

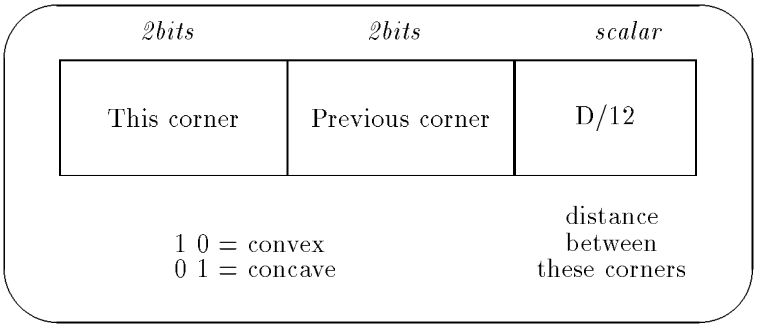

Figure 5.4: A typical input vector.

the input vector has roughly the same importance as the other components of the input vector shown in figure 5.4. It is obvious that, using this input vector, Alder and Cairngorm would only be able to differentiate between certain types of landmarks. In an enclosure of quadratic shape, for example, all four corners would be taken to be identical, in a non-quadratic rectangular enclosure two corners respectively would be taken to be the same.

Figure 5.5 shows the result of one experiment, using this vector: the robot's task was to recognise corner H. The way this and similar figures have to be read is this: the length of the bars shown at each location denotes the Euclidean distance $||\vec{\kappa} - \vec{\kappa_m}||$ between the excitation vectors produced at the corner that is to be recognised and the particular location. If this distance is big (that is, the bar in the diagram is high), the marked location and the current location are very dissimilar, if the distance is small, they are similar. Subsequent bars denote subsequent rounds around the enclosure. In figure 5.5 one can see that the robot identifies corner H after about three rounds without error, and that all other corners can be differentiated from corner H without problem.

Not all corners can be uniquely identified, however! Figure 5.6 shows that corners C and F get confused. A closer look at the input

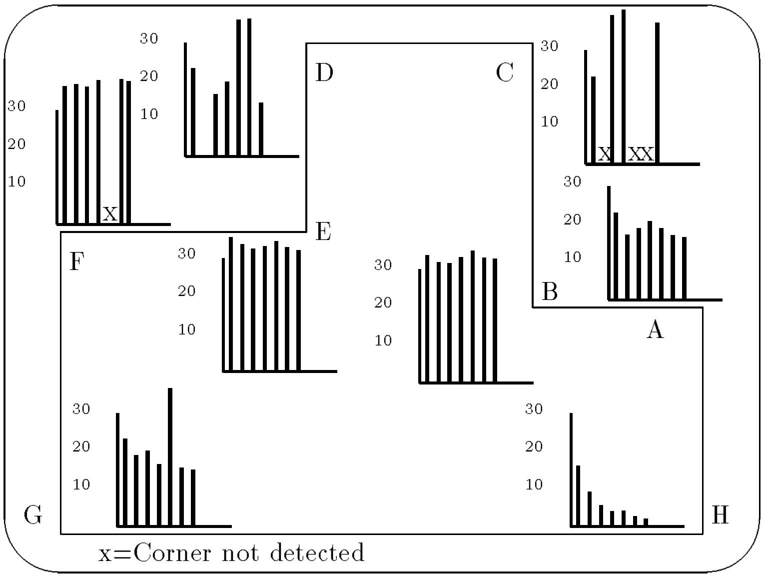

Figure 5.5: Recognising corner H, looking at one previous corner.

vector used (shown in figure 5.4) explains this: corners C and F look the same in the input vector, it is therefore not surprising that they evoke similar excitation patterns on the network. The same problem occurs when the task is to recognise corner B. Here, for the same reason, the robot confuses corners B and E (see figure 5.7).

The obvious answer to this problem is to increase the information contained in the input vector. The next input vector I used to train the network is shown in figure 5.8.

This vector contains information not only about the last corner encountered, but also about the corner before that one. But even this is not sufficient to differentiate between corners B and E, this can be seen in figure 5.10.

Only when information about the current as well as three previous corners is given to the net corners B and E look different in the input vector (see figure 5.9) and the robot is able to distinguish between them (figure 5.11). In both cases it is easy for the robot to recognise

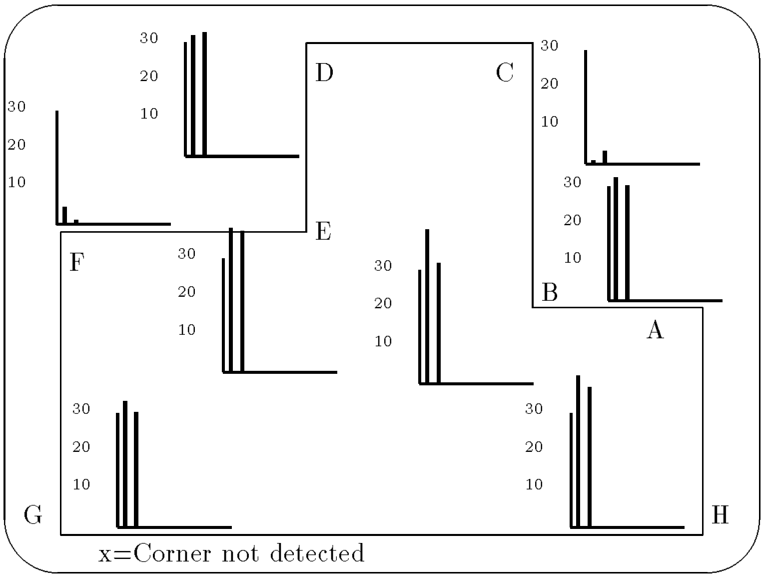

Figure 5.6: Recognising corner F, looking at one previous corner.

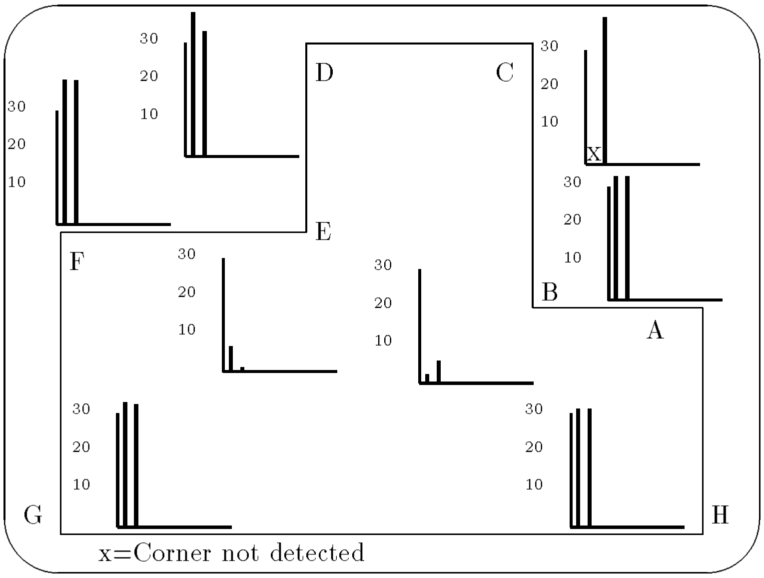

Figure 5.7: Recognising corner B, looking at one previous corner.

This Corner	Corner at time t-1	Corner at time t-2	Distance travelled betw. t & t-1

Figure 5.8: Extended input vector.

This Corner	Corner at t-1	Corner at t-2	Corner at t-3	Distance travelled betw. t & t-1

Figure 5.9: The input vector, even further extended.

corner H.

The longer the robot moves around in its enclosure, the clearer the excitation patterns on the map, and the smaller the Euclidean distance between $\vec{\kappa_m}(t)$ and $\vec{\kappa_m}(t+1)$. This is shown in figure 5.12.

In conclusion, this is what happened: having had sufficient time to explore the environment, the robot was able to recognise particular corners that had been marked by the experimenter, provided this was theoretically possible. In cases where the input vector presented to the self-organising net contained insufficient information, Alder confused corners that looked alike, simply because their input vectors were identical. This is not surprising. The robot's ability to recognise the marked corner reliably increased with experience, $||\vec{\kappa_m}(t) - \vec{\kappa_m}(t+1)||$, the difference between excitation of the net at the marked corner at

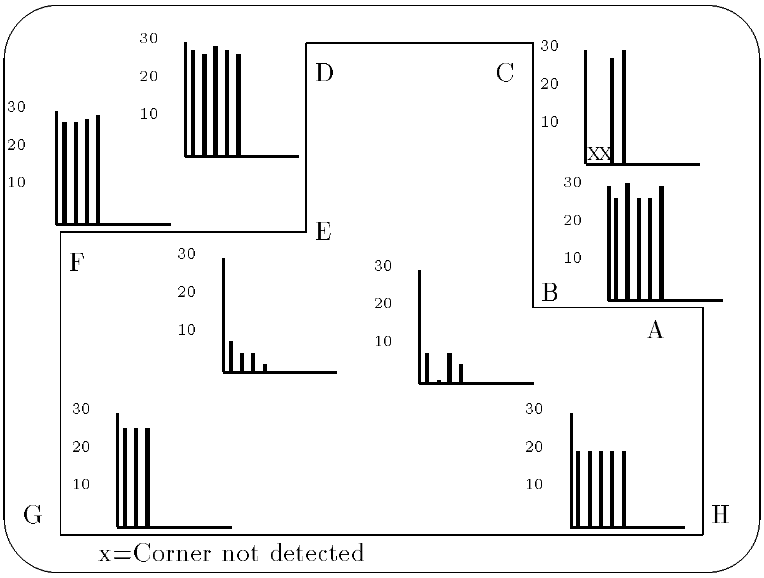

Figure 5.10: Recognising corner B, looking at two previous corners.

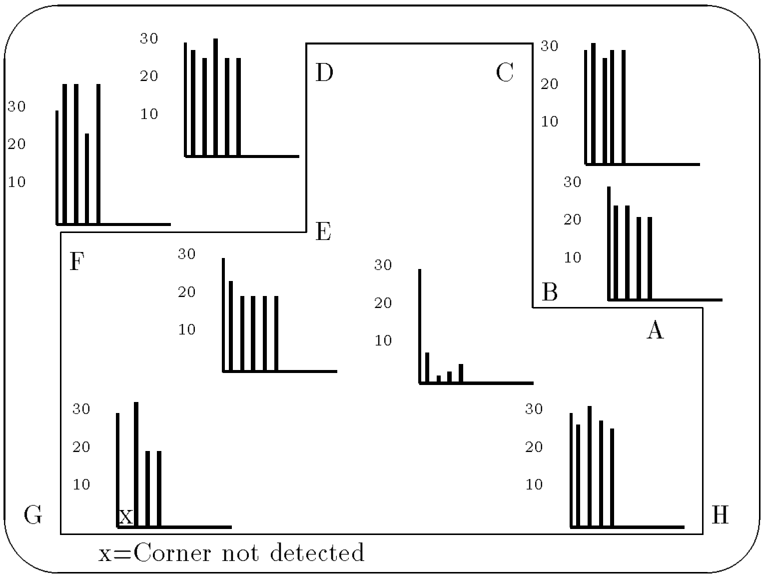

Figure 5.11: Recognising corner B, looking at three previous corners.

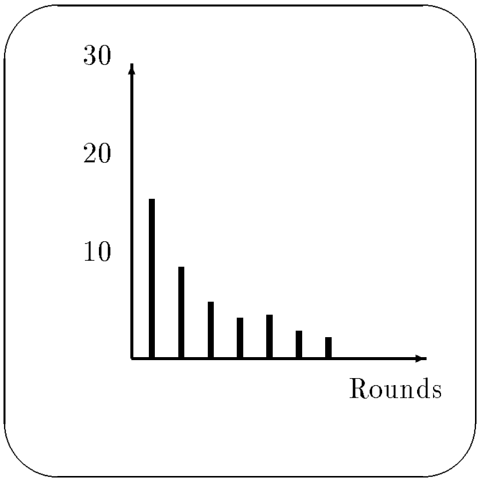

Figure 5.12: $||\vec{\kappa_m}(t) - \vec{\kappa_m}(t+1)||$ at corner H.

time t and excitation at the marked corner at time t+1, decreased continuously. It eventually became zero.

5.2.2 Discussion of this experiment

This experiment has shown that it is possible to use self-organising feature maps as internal representations of the 'landmark space' of a robot, and to use them for location recognition.

Alder has used a one-dimensional network of fifty units for this purpose, however it is not clear that this is the only possible network that could be used. Will one-dimensional networks be sufficient in all cases? Will a smaller network fail in the task? These are questions that are not directly addressed in this thesis (see also section 7.2), however, there are some guidelines.

A rule of thumb to estimate the required size of the net is the number of distinct features in the world that have to be mapped onto the SOFM. If all distinct features are to be mapped onto distinct, non-overlapping regions of the network, then a one-dimensional network needs at least $(2\delta + 1) \cdot \rho$ units, δ being the size of the neighbourhood region within which weight vectors are updated, and ρ being the number of distinct features to be mapped onto the map (this would mean forty units would be needed in this particular example, using a δ of two and having eight landmarks in the enclosure).

One criterion for determining the required dimensionality of the network is the dimensionality of the world the agent is to operate in. In Alder's and Cairngorm's case the world is effectively one-dimensional, because the robots follow the wall of their enclosure to generate the input signals to the SOFM. [Hertz *et al.* 91, pp.239f] give the example of a robot arm moving in three-dimensional space, avoiding obstacles, and suggest that "here a three-dimensional output array would clearly be appropriate". [Ritter 88, sect.9.1] discusses the use of SOFMs to learn the required torque for feedforward control of a simulated robot manipulator (i.e., determining the inverse kinematics and dynamics of the manipulator). This simulated manipulator has a two-dimensional workspace and the dimensionality of the network used is two-dimensional as well (15 x 24 units). [Kohonen 82b] presents a mapping of a one-dimensional input space (the response signals of twenty different bandpass filters to a single audio tone) onto a one-dimensional network of ten units. A topology-preserving tonotopic map develops.

Looking at the input vectors used in the experiments just described (on pages 133, 136 and 136), it becomes clear that the network can only usefully interpret incoming data if this data contains useful information about the enclosure. So, one might argue, the network is superfluous, all the information is the input signals anyway. However, whilst it is true that the relevant information *has* to be present in the input signals, the SOFM provides the following additional functions:

- Greater recognition reliability in the presence of noise: the odometry readings on Alder, for example, differ by as much as 20% for identical distances.

- Dimensionality reduction: here a multi-dimensional input space is mapped onto a one-dimensional network.

- A common interface between input signals (which can be changed, according to the experimental requirements) and output space (which stays unchanged, therefore allowing the training of the net and the interpretation of the excitation patterns to remain unchanged).

- A mechanism requiring little predefinition, thus following the RUR philosophy (see page 12), where any failure of the location recognition system can be ascribed to the input vector used (and not to the input vector *or* the method).

The mechanism for location recognition described above bears some resemblance to the way bees recognise places. As [Cartwright & Collett 83] have found in their experiments, honey bees (*Apis mellifera*) use nearby landmarks to guide their way to a food source.

> "[Bees] do not find their way using anything analogous to a floor plan or map of the spatial layout of landmarks and food source. The knowledge at their disposal is much more limited, consisting of no more than a remembered image of what was present on their retina when they were at their destination. ... Bees find their way, the experiments seem to say, by continuously comparing their retinal image with their snapshot and adjusting their flight path so as to lessen the discrepancy between the two."

The two aspects of navigation — location recognition and interpretation — are clearly contained in this statement. Unfortunately little is said about the latter: *how* bees lessen the discrepancy between stored and perceived image is not described. Alder's method of recognising locations shows similarities to the bees' way of achieving this: like bees, Alder does not generate a floor plan (a conventional map), instead it uses 'snapshots' (distinctive excitation patterns of the self-organising network as a response to the sensory input) to recognise locations. How this could be used for map interpretation is an open question, some ideas as to how it could be achieved are presented in section 7.2.

[Cartwright & Collett 83] also state that "the bee's guidance system is immune to a considerable amount of noise". To a small extent that could be observed in the experiments conducted here, too. As mentioned above, the revolution counter used as a crude means of measuring distance gives, for consecutive visits of the same corners, readings that vary by up to 20%. This means that two consecutive input vectors generated at that location are not identical; nevertheless Alder still recognises these locations as identical, largely due to the fact that the self-organising feature map produces a topological mapping in which similar, but not identical input stimuli excite neighbouring areas of the network. How far this immunity to noise can be taken is an interesting question and is one of the open questions discussed in section 7.2.

5.2.3 Summary

The Experiment at a Glance

Title of Experiment: Using landmarks for location recognition.

Robot used: Alder.

Sensors used: Two tactile sensors, mounted at the front of the robot, and a revolution counter for odometry.

Robot behaviour used: Hardwired wall following and obstacle avoidance behaviour.

Network used: Self-organising feature map, implemented as a ring of fifty units. Neighbourhood region ± 2.

Input vector used: Three different vectors in three related experiments, containing information about the sequence of detected landmarks as well as a crude distance measurement between the current and the previous landmark (see figures 5.4, 5.8 and 5.9).

Magic numbers used: Gain $\eta = 5.0$, decreasing by a factor of 0.95 every epoch (see equation 2.5). Neighbourhood region ± 2, constant over time. The goal location is identified if the Euclidean distance between target location and current location is less than 7.0 (this number 7.0 is then replaced by that difference plus 2.0).

Results and observations: Exploring an enclosure by following its walls, Alder trains the self-organising feature map using information about detected landmarks. The robot reliably recognises locations, after the network has settled in a stable state, provided sufficient information is given in the input vector to the network.

5.3 Using All Motor Actions for Location Recognition

With the experiment described in this section I tried to get away from having to predefine how convex and concave corners can be detected. Also, there was another point I tried to make. Sensing and acting are typically treated as separate functions in robotics. I believe, however, that sensing and acting are two aspects of the same function (the interaction between the robot and its environment), and that they therefore cannot be successfully analysed in isolation. The actions of a robot, just like those of a person, determine to a large extent the sensory signals it will receive, which will in turn influence its actions. Breaking this tight interaction into two separate functions leads, I believe, to an incorrect decomposition of the robot control problem. While particular features of a robot's sensors and actuators do play an important role in determining its performance, these effects cannot be determined by their separate analysis. Acting and sensing have to be seen together; neither acting nor sensing alone will make the agent succeed.

The input vector I have chosen for the experiments described below demonstrates this point: it contains no direct information about sensory input. The information it does contain is derived from the motor action commands of the robot controller, which are themselves influenced by the sensory signals received by the robot as a result of its actions. I chose to use information derived from the motor action commands of the robot controller, rather than from sensor signals, because they form a smaller set of signal types, they are much less subject to noise, but they still adequately characterise the interactions between the robot and its environment as it seeks to achieve its task — wall following in this case.

Again the robot was placed in an enclosure as shown in figure 5.1;

it then followed the wall using a preprogrammed wall following and obstacle avoidance behaviour. In other words, the robot is governed by its preprogrammed wall following behaviour which, of course, does use sensory information. The process of constructing the self-organising feature map is, however, independent of the wall following behaviour, it simply "looks" at the motor action commands issued by the controller as the robot performs this wall following task.

Every time a new motor action command is issued (i.e. every time the motor state changes) due to the wall finding or the obstacle avoidance behaviour becoming active, a motor action vector is generated. This is a nine bit vector which contains information about the state the motors were in until this change, and thus the direction the robot has been travelling up to this moment (forward, left or right)[4], as well as information about how long it was in this state (see figure 5.13). The last motor command is encoded in four bits, because two bits refer to each of the two motors, and two bits per motor were used to maintain non-zero input vectors in all situations. The thresholds for encoding the duration of a motor action are related to the speed the robot travels at ($3cms^{-1}$) and the dimensions of the enclosure. They are chosen so that all six "time bins" are selected approximately evenly. This motor action vector forms the input to the self-organising feature map.

Thus, from figure 5.13 we can see that no information concerning sensor signals is directly presented to the self-organising feature map. The only information available to the network concerns motor action commands.

It is the sequence of motor action commands that have been issued prior to the arrival at the particular physical location that enables

[4]Strictly speaking, they contain information about the last command from the robot controller. Whether this command was actually obeyed by the robot or not is not sensed.

Motor Action		**Duration**
Forward	01 01	00000 less than 0.9s
Left	01 10	00001 0.9 - 1.3s
Right	10 01	00011 1.3 - 1.7s
		00111 1.7 - 2.1s
		01111 2.1 - 2.6s
		11111 over 2.6s

Figure 5.13: The motor action vector.

Alder to recognise the location. To achieve this I used a system of seven independent, two-dimensional self-organising feature maps[5] working in parallel. The number seven was chosen because during one complete circuit round the enclosure about fifty input vectors are generated. The sampling theorem requires to sample at a rate of at least twice the highest frequency, say $\frac{48}{2}$, which allows to use input vector lengths that follow a geometrical series: 2,4,8,16. Adding in-between vectors of length 6, 12 and 24 results in seven distinct input vectors for seven independent networks. Each self-organising feature map consists of a network of twelve by twelve cells. The input vectors to each of these networks are different, but all are built from motor action vectors as shown in figure 5.13. By combining 2, 4, 6, 8, 12, 16, and 24 of these basic motor action vectors, seven self-organising feature map input vectors were formed which correspond to increasingly longer histories of the robots motor action changes (see figure 5.14).

The experiment was conducted like this: the robot was set to wall follow its way around the enclosure. Every time a new motor action command was issued as a result of the built-in wall following or obstacle

[5] I chose *two-dimensional* networks to begin with in order not to restrict the system, as I could not know whether it would work at all.

avoidance behaviour of the robot, a motor action vector as described below was generated. This vector, together with the respective number of previous motor action vectors was presented to each of the seven self-organising feature maps. After a sufficient time, about five times round the enclosure, these feature maps had organized themselves into stable structures well correlated with the patterns of experience of the robot. After this training period the excitation patterns of all seven networks at a particular location (the *target patterns*) were stored. All subsequent sets of seven excitation patterns generated by new input vectors (*object patterns*) were then compared to the set of seven target patterns. This was done by computing the Euclidean distance (or, alternatively, the city-block distance) between pairs of target and object patterns (see equation 5.2). If the distance values between each of the seven pairs of object and target patterns are less than a threshold defined for each pair, the robot is taken to have arrived back at the target location and thus to have recognised the previously stored location.

In mathematical terms, the system works as follows:

1. Compute the output o_{xyj} of each cell at position (x, y) of each network j:

$$o_{xyj} = \vec{w}_{xyj} \cdot \vec{i}_j \quad j = 1, ..., 7, \tag{5.1}$$

   where $\vec{w}_{xyj}$ is the individual weight vector of cell(x, y) of network j, and $\vec{i}_j$ is the input vector to network j.

2. Compute the distance between target pattern and corresponding object pattern:

   (a) Either the Euclidean distance is chosen,

$$d_j^e = \sqrt{\sum_{x=1}^{12} \sum_{y=1}^{12} (o_{xyj} - o_{xyj}^T)^2}, \tag{5.2}$$

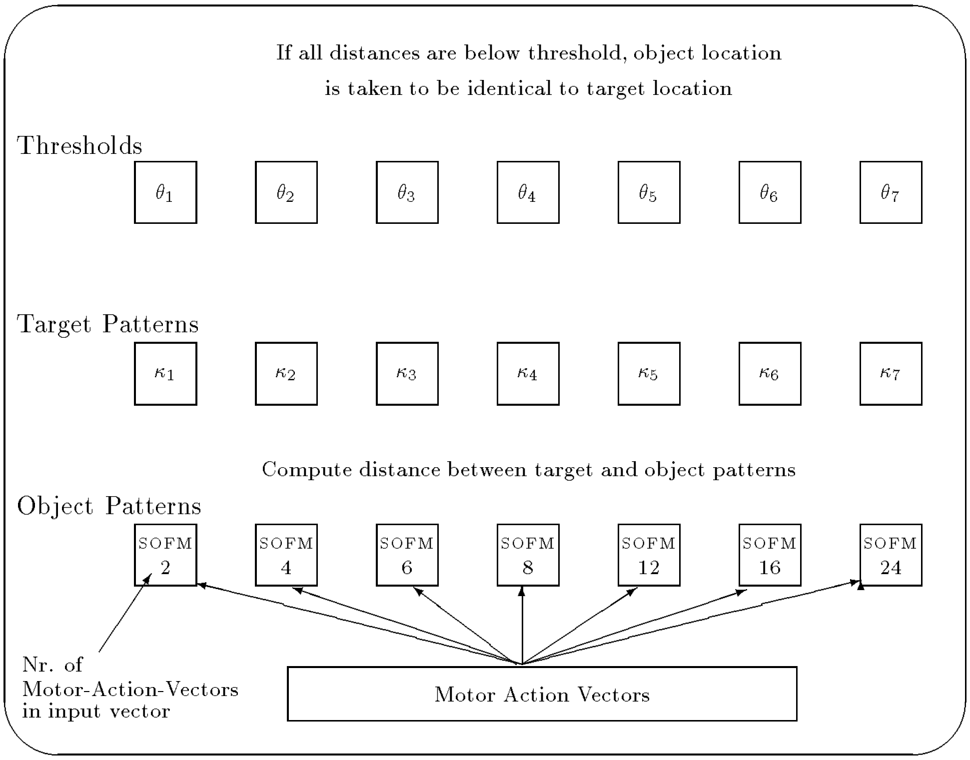

Figure 5.14: The system used for location recognition.

where o_{xyj} is the output value of cell(x, y) of the object pattern j, and o^T_{xyj} is the output value of cell$_{xy}$ in target pattern j.

(b) Or, the city-block distance is computed

$$d_j^{cb} = \sum_{x=1}^{12} \sum_{y=1}^{12} |o_{xyj} - o^T_{xyj}|. \tag{5.3}$$

3. Determine whether object location and target location are identical: If $d_j < \Theta_j$ for all j=1 to 7 then object location and target location are taken to be identical. Θ_j is the threshold for network j. For the experiments reported here the Θ_j were chosen by the designer so that patterns obtained at target location and

patterns obtained at other locations could be separated. The values will, of course, depend on the distance metric chosen.

5.3.1 Experimental Results

The results[6] were these: in the experiments conducted, the robot's task was to identify three particular corners in the enclosure shown in figure 5.1: corners H, E and F. To do this, the set of seven excitation patterns of the seven self-organising feature maps at the target corners were stored; all subsequent sets of excitation patterns were then compared to these by computing the Euclidean distances between respective pairs of patterns in the set of seven.

Provided a suitable[7] set of thresholds was used, the robot recognised corner H four times in the subsequent five rounds, and corners E and F in five out of the five times. At no time was a non-target corner erroneously taken to be the target corner.

Instead of using the Euclidean distance to estimate similarity between excitation patterns, the city-block distance can be used with identical results. City-block distance is computationally cheaper than Euclidean distance. Instead of using two-dimensional networks, one-dimensional networks can also be used. This considerably reduces the computational cost of the scheme. I performed experiments using one-dimensional networks (see appendix C), however because the scheme was still computationally expensive I chose to simplify it before using it in real time on Cairngorm (see section 5.4).

[6]In this experiment computation was done off-line, using data obtained from the robot.

[7]These suitable thresholds were hand-crafted — the fact these many thresholds had to be determined was the reason why I later tried a simpler scheme, see section 5.4.

5.3.2 Discussion

There are two main conclusions that can be drawn from the experiment discussed in this section. First, the claim that sensing and acting are closely coupled is confirmed by the success of the robot in recognising locations based upon the sequence of motor activity which leads to arrival there. The "sensor" being used to provide features on which the recognition is based is actually the behaviour of the robot. Choosing an input vector that contains no direct information about sensory signals makes the system independent of the actual sensors used. Whether tactile, ultrasonic, infrared or other sensors are used: the location recognition system stays the same. Secondly, with this approach the features to be identified by the self-organising feature maps are spread out over time. This contrasts with the experiments on competence acquisition mentioned in chapter 4 in which the robot controller learns instantaneous reactions to sensor stimuli.

It is not clear that two-dimensional networks have to be used. I have also used one-dimensional networks of twenty-five units each, with the result that corners H and E were identified as in the two-dimensional case, but that corner F was once confused with another corner, as well as identified correctly five times, as before ([Nehmzow *et al.* 91b]). It seems, therefore, that at the expense of some reliability, one-dimensional networks could be used, which still makes this scheme seven times as costly in terms of computational load as the previous experiment (and the following one, which also uses one one-dimensional network).

The biggest problem with this scheme is to determine suitable thresholds. Ideally, if true autonomy is to be achieved, these thresholds should be determined by the robot, not by the designer. Even though they could probably be learned in a supervised learning scheme (a classification system might be able to learn the necessary thresholds), finding the right set of thresholds in a space with seven degrees of freedom is very difficult and time consuming, finding a single threshold is far easier (again, it could be determined through a supervised learning scheme). As this experiment has shown, it *is* possible to manually find a suitable set of thresholds, but it was not a straightforward and easy thing to do.

The biggest task of the networks was to distinguish between 'interesting' motor actions (those that are performed at landmark locations) and uninteresting ones (the motor actions that are a result of ordinary wall following behaviour). Essentially the seven networks performed a kind of frequency component analysis on the pseudo-periodic sequence of input vectors generated as the robot completed circuits of its enclosure. In the next section I will present an architecture that performs the same task — distinguishing between interesting and uninteresting features in the world — but uses only one network, which has the advantage of a lower computational load as well as fewer magic numbers (i.e., thresholds).

5.3.3 Summary

The Experiment at a Glance
Title of Experiment: Using all motor actions for location recognition. **Robot used:** Alder was used to obtain the data, the computation was done on a Sun3 workstation. **Sensors used:** Two tactile sensors, mounted at the front of the robot, and the clock of the controller. **Robot behaviour used:** Hardwired wall following and obstacle avoidance behaviour. **Networks used:** Seven independent, two-dimensional self-organising feature maps of 12x12 cells each. **Input vectors used:** Seven input vectors for the seven networks, containing information about the motor actions the robot performed, looking back in history to varying amounts (see figures 5.13 and 5.14). **Magic numbers used:** Gain $\eta = 0.2$; seven independent thresholds for location recognition (manually determined and difficult to find). **Results and observations:** Using a set of seven independent self-organising feature maps, it was possible to recognise locations by merely using information about all motor actions the robot performed. One major task of the networks therefore was to separate 'interesting' from 'uninteresting' motor actions.

5.4 Using "Significant Motor Actions" for Location Recognition

In section 5.2 I used detected corners, i.e. landmarks, to construct the input vector that trained the self-organising feature map. If sufficient information was given in the input vector, the robot was able to recognise locations without difficulty.

In order to achieve the same result without predefining what a landmark looked like, I then used a system of seven independent self-organising feature maps (previous section) and input information solely based on the motor actions of the robot to recognise locations. The idea followed the RUR philosophy of avoiding predefined knowledge: no feature detector was needed, features were autonomously recognised through the system of networks. A drawback of this approach was that the scheme was at least seven times as expensive in terms of computational load as the previous one (which in our case meant that it was never tried in real time on the robot; instead, real data was analysed off line on a more powerful computer).

The major difficulty, however, is to determine a set of seven (instead of one) appropriate thresholds that allows the robot to identify and distinguish locations. The question, therefore, was: could this scheme be altered so that it would still perform the same task, i.e. distinguish between "interesting" and "uninteresting" locations, but require fewer thresholds and, as a bonus, be computationally cheaper?

As [Barlow 89] argues, *redundancy* is what drives unsupervised learning in the biological world:

> "Redundancy is the part of our sensory experiences that distinguishes [information] from noise; the knowledge it gives us about the patterns and regularities in sensory stimuli must be what drives unsupervised learning. . . . How can something be recognised as new and surprising if there is no preexisting

knowledge about what is old and expected?"

The difference between the maximum rate of presentation of usable information and the actual rate at which information is delivered is what can be exploited by the animal for learning. There are many ways of finding regularities in the sensory input; the simplest are mean, variance and covariance. I have used the mean for the experiments described in this section. By introducing the idea of *significant motor actions* it was possible to train a self-organising feature map and use the excitation patterns of the map to recognise locations, without having to set critical thresholds, and at low computational cost.

I define significant motor actions as those motor actions whose duration exceeds the duration of the average motor action. Similar techniques can be found in biology: pigeons, for example, extract the changes in air pressure generated by changes in altitude of a few feet by ignoring the total strength and only measuring differences around some mean, [Gould 82]. The important difference between the experiments described here and the experiments described in the previous section is that here input vectors are only generated when a *significant* motor action is performed, not when *any* motor action is performed.

As in the very first experiment discussed in section 5.2, I have again used a one-dimensional, ring-shaped self-organising feature map of fifty cells with a neighbourhood region of one node on either side. The robot used was Cairngorm. Cairngorm's controller consists of three independent behaviours: an obstacle avoidance, a wall seeking and a mapbuilding behaviour. Obstacle avoidance and wall seeking behaviour were preprogrammed and fixed in the experiments described in this chapter — later I combined competence acquisition and mapbuilding (this will be described in chapter 6). Obstacle avoidance was achieved by defining the connections between sensors and actuators as shown in figure 4.9; wall seeking was triggered if the robot hadn't received any sensor signal for a preset amount of time. In this case the robot was programmed to turn right until either the left or the right whisker touched something.

5.4.1 Experiments

As the robot makes its way around its enclosure, wall following and avoiding obstacles, the average duration for turn actions is computed. If a turn action occurs which takes longer than the current average (a ‘significant turn action’) an input vector is constructed (see vector 1 in figure 5.16), fed into the self-organising network and the modification to the weight vectors of the net as defined in section 2.4.2 is performed. After about two to three times round the enclosure the one-dimensional ring develops a stable enough structure to be used to recognise particular locations.

When the robot arrives at the location that is to be recognised, a pushbutton switch is pressed and the current excitation pattern of the network is stored. This stored pattern is then compared with all subsequent excitation patterns. If the pattern at the current location of the robot is found to be sufficiently similar to the stored one, the robot indicates that it has arrived back at the home location. “Sufficiently similar” here means that the city-block distance between the excitation vector at the current location and the excitation vector at the stored location is less than $\frac{1}{3}$ of the average distance. $\frac{1}{3}$ is an arbitrarily chosen value which gives a high safety margin to avoid confusion with other locations. If this value is too big, other locations might be confused with the target location, and if it is too small, even the target location might not be recognised. The city-block distance ϵ is given by:

$$\epsilon = \sum_{k=1}^{50} |o_{s_k} - o_{c_k}|, \tag{5.4}$$

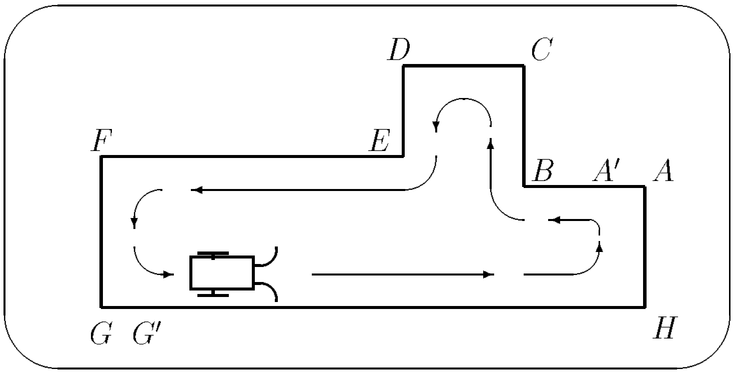

Figure 5.15: The experimental enclosure for location recognition.

where $\vec{o}_s$ is the stored excitation pattern, and $\vec{o}_c$ is the current excitation pattern. Note that here the fact that self-organising feature maps produce topological mappings is not exploited, as it is not the index of the most excited cell that is used for similarity measurement, but the excitation of the whole network. In the early stages of the self-organising process this offers advantages, because even if the network has not settled in a state where slightly different inputs belonging to one class will excite the very same unit, they will at least excite similar units. This can be detected by taking into account the excitation of the whole network. Once the network is stable, index information alone should suffice to detect similar input vectors ([Lampinen 91]). This has not been established experimentally, however.

5.4.2 The Experimental Results

The robot is placed in a simple enclosure, containing right-angled convex and concave corners as well as straight walls. Figure 5.15 shows the layout of the enclosure. The letters indicate locations where a turn action usually exceeds the average duration and therefore where an input vector to the self-organising feature map is usually generated.

	2 bits	*2 bits*	*3 bits*	*2 bits*	*3 bits*
Input vector 1:	TA_{t_0}	$TA_{t_{-1}}$	Δt_{0-1}	$TA_{t_{-2}}$	Δt_{1-2}
Input vector 2:	TA_{t_0}	$TA_{t_{-1}}$	Δt_{0-1}		
Input vector 3:	TA_{t_0}	$TA_{t_{-1}}$			

Key:

Turn Action,	$TA_t =$	0 1 for left
		1 0 for right
Time Difference,	$\Delta t =$	1 1 1 for $800 < \Delta t$
(in robot time units)		0 1 1 for $400 < \Delta t < 800$
		0 0 1 for $200 < \Delta t < 400$
		0 0 0 for $\Delta t < 200$

Δt_{0-1} is the time between Turn Actions at t_0 and at t_{-1}.

Δt_{1-2} is the time between Turn Actions at t_{-1} and at t_{-2}.

Figure 5.16: Input vector definition for the self-organising controller.

As can be seen from input vector 1 in figure 5.16, the information used as input to the self-organising network consists of the type of the current turn action (right or left), and the types of the previous two turn actions, together with the elapsed time between each pair.

In order to investigate the performance of this location recognition scheme I used two other input vectors, see input vector 2 and 3 in figure 5.16, each one containing less information than the previous one. Input vector 2 consists of the types of the current and previous turn actions and the elapsed time between them, and input vector 3 consists of just the types of the current and previous turn actions. The experimental setup when using each of these three kinds of input vector was in all other respects the same.

Figure 5.17 shows the location recognition results obtained using

Location	Identifications	Total no. of Visits
H	3	4
B	3	3
C	3	3
D	3	3
E	3	3
F	3	3
G	3	3
G'	2	4
A	3	3
A'	3	3

Figure 5.17: Location identification using input vector 1.

input vector 1 (see figure 5.16)[8].

The letters refer to the locations shown in figure 5.15. Except for A' and G', they correspond to the corners of the enclosure. This is because turn actions that take longer than average are produced at these places. The locations A' and G' are produced because the robot typically takes two long turn actions to negotiate these particular places, coming from corners A and G respectively.

The one missed recognition at corner H occurred early on in the run and was due to the network not having settled down enough for a successful recognition to be registered. The two misses at G' occurred because on these occasions the robot got round corner G with one turn action, and so the region of the self-organising feature map corresponding to an input vector having G', G, and F was not excited

[8]Figures 5.17, 5.19 and 5.21 indicate how often the robot identified the respective corner as being identical to the goal location, figures 5.18, 5.20 and 5.22 show which corners were erroneously taken to be identical with the goal location.

Goal↓	A	A'	B	C	D	E	F	G	G'	H
A										
A'									X	
B										
C										
D										
E										
F										
G										
G'		X								
H										

Figure 5.18: Confused locations when using input vector 1.

by any input vector.

Figure 5.18 indicates the pairs of locations which were confused during the run. This figure is symmetrical about the diagonal, as one would expect, however figures 5.20 and 5.22 are not. This is due to the occurring variation of input vectors generated en route towards the respective locations, making for example corner G "look like" D, but not D "look like" G (figure 5.22).

In this case it was only locations A' and G' which were confused. This occurs because the input vectors at these locations (built from H, A, A' and F, G, G' respectively) are similar, and thus excite the self-organising network in the same way.

Figure 5.19 presents the location recognition results when using input vector 2. As can be seen, there is a degradation in performance with missed locations occurring at D, F, and G. The failure to recognise location F here was due to an input vector not being generated at location E, thus leading to an 'odd' vector at F. This occurred only

Location	Identifications	Total no. of Visits
H	3	3
B	3	3
C	3	3
D	3	4
E	3	3
F	3	4
G	4	5
G'	3	3
A	3	3
A'	3	3

Figure 5.19: Location identification using input vector 2.

once and was the result of the inevitable variation in actual behaviour experienced when using real robots (even simple ones) in a real environment. This was, however, a one-off event and never observed to occur again.

The degradation in performance can be seen more clearly in figure 5.20, which presents the pairs of confused locations. The bracketed pairs denote occasional confusions.

Reducing the amount of information in the input vector still further, and using input vector 3, I obtained the results presented in figures 5.21 and 5.22. Here it becomes obvious that the robot is still able to recognise locations, but its ability to distinguish certain pairs of them is significantly diminished.

As one might expect, the performance of the whole system depends on the information put into the self-organising network. This is the same observation I made earlier on, when using landmark information for location recognition. If the input vector contains insufficient in-

Goal↓	A	A'	B	C	D	E	F	G	G'	H
A		X			(X)			X	X	
A'	X							X	X	
B										
C										
D										
E										
F										
G	X	X			(X)				X	
G'	X	X						X		
H										

Figure 5.20: Confused locations when using input vector 2.

Location	Identifications	Total no. of Visits
B	3	3
C	3	3
D	2	2
E	2	2
F	2	2
G	2	2

Figure 5.21: Location identification using input vector 3.

Goal↓	A	A'	B	C	D	E	F	G	G'	H
A		X			X			X	X	X
A'	X				X			X	X	X
B						X				
C							X			
D	X	X						X	X	X
E			X							
F				X						
G	X	X							X	X
G'	X	X						X		X
H	X	X						X	X	

Figure 5.22: Confused locations when using input vector 3.

formation, then reliable location recognition is impossible. From the results presented above one can see that reducing the information content of the input vector does not affect the robot's ability to recognise a non-wall type environmental feature (corners in this case), but it does affect its ability to differentiate some pairs of such locations. The less informative the input vector, the more locations look the same to the robot. This is not surprising: people also get confused at times, for example in big office buildings where different floors look identical (unless, of course, they look more closely for landmarks such as room numbers).

5.4.3 Discussion of this experiment

The experiments described in section 5.3 used a set of seven self-organising two-dimensional networks to get the robot to reliably recognise (and distinguish between) locations[9]. Yet in the scheme discussed in this section I only need one one-dimensional network (ring) to achieve similar performance results. This effect can be explained by observing that the new scheme works by filtering the sequence of motor commands so that only those *not* generated by following a straight wall are used to build input vectors to the self-organising structure, whereas in the previous scheme the set of seven networks had to do this filtering work implicitly.

Another way of viewing this is to say that the sequence of motor commands produced as the robot wall follows its way around the enclosure contains two kinds of structure. One kind occurs at a high frequency and is produced by the wall following actions. The other kind has a lower frequency and is produced by the corners of the enclosure. It is this second type of structure in the motor action commands that contains information about significant structure in the robot's environment (corners in this case); the first type merely reflects the fact that the robot's environment has straight walls in it — a rather less useful piece of information for location recognition.

The architecture discussed in section 5.3 is computationally at least seven times more expensive than the one discussed here, but a bigger problem is this: seven appropriate thresholds for the seven networks have to be found in order to recognise a location. Here this number is reduced to one.

To make finding this threshold even easier and more independent from specific knowledge, I used an automatic scheme for determining it. When I first started to experiment, the turn action time threshold, used to distinguish wall following actions from other actions, was set by hand. I arrived at the particular threshold value by carefully observing

[9]At the cost of smaller reliability, one-dimensional networks could have been used.

the robot's behaviour and choosing it such that it would differentiate between motor actions performed at a 'significant' location (usually a corner) and those performed elsewhere (while wall following). Later I implemented the average calculation, thus removing the need for this magic number to be set by hand (the new magic number I have introduced by doing this, namely the the proportion by which the duration of a motor action has to exceed this average to become a significant action, turns out to be not very critical; therefore it is far easier to set). Using this simple device in the robot means that it does not need any predefined knowledge about thresholds and significant motor actions, it finds out for itself.

I could not have used a similar scheme to determine the seven thresholds for the networks used in the previous experiment, because the task to 'reject' certain locations as insignificant is shared between the nets. Some of the seven networks would let an insignificant location 'pass' (i.e., the difference between the excitation pattern at the current location and the stored location is smaller than the threshold for that particular network), others wouldn't. There are situations where six out of seven networks (erroneously) 'identify' a location as the stored one, but the seventh network rejects it. This indicates that the set of seven thresholds has to be carefully chosen, which makes it far more difficult to find appropriate values than it is in the experiment discussed here.

Obviously, in this experiment the robot is only able to recognise significant locations. Any location between those significant locations cannot be recognised, because no input vector is generated there. Theoretically, the seven-network architecture can recognise any location where a motor action is performed, even parts of a straight wall. Whether this holds in practice, I have not investigated.

A similar threshold value is used in the comparison between the stored ('home') excitation pattern and all subsequent patterns. Once again, I started by determining the required value for this empirically and setting the value in the program by hand. But having devised the successful averaging mechanisms to set the turn action time threshold I decided to try a similar device for the comparison threshold. Instead of using a fixed maximal city-block distance between the excitation vectors at stored and current location, the robot uses the average of all those distances: once the distance is smaller than $\frac{1}{3}$ of the average the robot assumes it has arrived back at the stored location. The interesting aspect about this is that $\frac{1}{3}$ is no absolute value, but a relative one. Thus it is less prone to errors due to changes in the robot or environment; a relative value is also less critical to set (instead of $\frac{1}{3}$ I could have chosen $\frac{1}{2}$ or $\frac{1}{4}$, making the robot less or more selective).

In this experiment, the average duration of a motor action was computed over all motor actions ever executed by the robot. This means that the threshold for distinguishing significant from insignificant motor actions will change more at the beginning of an experiment, and less towards the end. Normally, this is a desired effect. Being more unstable at the beginning, the threshold gradually settles to a stable value and only persistent, long-term changes in the robot or the environment will make this value change. This property has proven to be useful for example when the robots' motor actions become slower due to decreasing battery charge. Although each turn of Cairngorm then takes longer, the average duration of a motor action also increases, so that significant locations are still reliably detected.

It is conceivable to keep the robot more 'alert' to changes by averaging over a time window of limited, rather than infinite length. The length of this time window could be modulated by a novelty detector; if a new situation is detected the time window could be shortened to make the threshold finding mechanism more susceptible to the change.

Whether this would be a good or a bad thing, however, depends upon the particular task and the particular conditions.

5.4.4 Summary

The Experiment at a Glance
Title of Experiment: Using significant motor actions for location recognition. **Robot used:** Cairngorm. **Sensors used:** Two tactile sensors, mounted at the front of the robot, and the clock of the controller. **Robot behaviour used:** Hardwired wall following and obstacle avoidance behaviour. **Network used:** A ring-shaped self-organising feature map of fifty units. **Input vector used:** Three different vectors in three related experiments, containing information about the sequence of significant motor actions performed by the robot, as well as information about the time that elapsed since the last significant motor action (see figure 5.16). **Magic numbers used:** Gain $\eta = 0.2$ (not critical); three time thresholds for constructing the input vector (see figure 5.16), related to robot speed and chosen so that all time bins are filled roughly equally; a location is identified if the Euclidean distance between its excitation pattern and the target excitation pattern is less than $\frac{1}{3}$ of the average Euclidean distance between other excitation patterns and the target pattern (this $\frac{1}{3}$ is not a critical value). **Results and observations:** Again, reliable location recognition was achieved; in this experiment the computational load was dramatically reduced, compared to the previous experiment (section 5.3), by introducing *significant motor actions*. These are motor actions that take longer than an average motor action.

5.5 Summary and Discussion of the Location Recognition Experiments

In three sets of experiments Alder and Cairngorm have autonomously acquired the ability to recognise locations within their enclosure they had visited before. The experiments were similar in that they all used self-organising feature maps to represent the experience space of the robots. They differed in the input information used to train the networks.

The first set of experiments used information about landmarks to train the network. These landmarks were detected by the robots by using a predefined wall following behaviour: turn actions exceeding certain thresholds denote concave or convex corners, the landmarks.

The second set of experiments still used the predefined wall following behaviour, however the information used for self-organisation was derived from the motor actions the robots performed. Every motor action was taken into account, and a set of seven independent self-organising feature maps was used to separate significant from insignificant motor actions, and to recognise locations.

The last set of experiments combined aspects of both previous experiments: it used just one network, like in the first experiment, and it used significant motor actions as inputs to the self-organising feature map, as in the second experiment. Unlike the previous experiment, significant motor actions were not detected by a set of seven independent networks, but by comparing the duration of each motor action with the average duration of motor actions. Every motor action that exceeded this average was denoted a significant one and used as input information to the self-organising feature map.

Obviously, I have shown that in the particular experimental setup I have used it is possible for mobile robots to recognise places they have visited before, using self-organising feature maps. However, two more interesting features can be seen here. Firstly, in the earlier experiments in motor competence acquisition, learning was only possible because immediate feedback about success or failure of a particular action was available to the controller. Such information is not always available, sometimes a sequence of actions has to be performed before success or failure are established. In the location recognition experiments, the robots recognised locations by looking at sequences of landmarks or sequences of motor actions. A particular sequence of these events was mapped onto a particular area of the self-organising feature map and could thus be identified. This mechanism could be very useful for future experiments, where an immediate feedback function is not available. Secondly, the self-organising feature map can be used to perform a dimensionality reduction by only taking into account the unit responding most strongly. This is a great advantage for robots with limited computing power. Every dimensionality reduction causes loss of information, however the self-organising feature maps perform this reduction in a way that will keep the most important information: they detect the dimension in input space with the highest variance (which has the highest information content, unless the environment is very noisy) and project the input space onto the output space along that dimension. "The map ... will become aligned along those directions of the signal distribution where the variance is most significant. Moreover, the map attains a format which conforms to the signal distribution", [Kohonen 82a]. Self-organising feature maps provide a suitable means to reduce the dimensionality of high dimensional sensor input spaces, and are thus able to abstract this input information.

Having presented a self-organising controller that enables mobile robots to autonomously acquire basic motor competences (chapter 4), and having discussed ways of using self-organising feature maps for location recognition, using hardwired motor competences of the robot, it seems natural to combine the two sets of experiments and build a robot controller that enables the robot to first autonomously acquire motor competences, and then use these to train a self-organising feature map for location recognition. I am going to discuss such an architecture in the following chapter.

Chapter 6

Staged Learning

> All learning tends to utilize and build on any earlier learning, instead of replacing it, so that much early learning tends to be permanent; and finally, that the learning of the mature animal owes its efficiency to the slow and inefficient learning that has gone before, but may also be limited and canalized by it.
>
> [Hebb 49]

6.1 Introduction

Growing from "weak to strong" ([McGonigle 91]) is one of the key issues in animal learning; this should apply to intelligent robotics as well. What is meant by this phrase is that competences once acquired by the animal or the robot are utilised in the learning that occurs subsequently to acquire further competences. The whole learning process is seen as a staged process with genetically programmed steps, built upon each other and depending on each other. There seems to be biological evidence to support this view. [McGonigle 91]:

> "[A] staged view of human cognitive growth has received strong support from research in the neurosciences. [Thatcher *et al.* 87], using EEG phase and coherence measures, have picked up a pattern of growth spurts and consolidations in the development of cortico-cortico connections between hemispheres in human brain development seen by them

> as evidence of genetically preprogrammed additions of new neural systems. If so, the results imply that many of the important cognitive leaps forward in development are the main consequence of such changes in the nervous system and are thus by 'design' and not a consequence of earlier antecedent behaviours."

This says that being intelligent cannot be learned, but is the result of an appropriate (genetic) "program". [McGonigle 91] describes the process of incrementing intelligent systems by design as a process along two orthogonal trajectories. The vertical trajectory is the direction of functional hierarchy — this describes the potential abilities of the system, such as for example obstacle avoidance or mapbuilding — the horizontal one is the actual implementation of each level of competence. In this sense the self-organising controller discussed in chapter 4, used for motor competence acquisition, is the actual implementation (horizontal trajectory) of motor competences (vertical trajectory). The same applies to the self-organising feature map (horizontal) and competence of location recognition (vertical). The traversal along the vertical axis in biological systems seems to be genetically programmed; in this chapter I present an experiment in which a similar traversal along the vertical axis towards higher levels of competence is executed, based on a pre-programmed (designed-in) decision rule that triggers it.

In chapter 4 I have discussed experiments concerning the autonomous acquisition of basic motor competences by mobile robots. Through trial and error, the robots were able to associate task-achieving motor actions with particular input stimuli. Because of their learning capability, the robots were also able to adapt to circumstances changing in unforeseen ways. The learning described in chapter 4 was of the "stimulus-response" type: an immediate response to an external stimulus was learned, with no intermediate stages of learning involved.

Then, in chapter 5, I presented a self-organising controller that enabled the robots to construct a representation of their environment and use this representation to recognise locations in their world which they had visited before. The input stimuli used to develop the self-organising feature map in the controller were either based on landmark recognition (i.e., sensory information, see section 5.2) or on the motor actions performed by the robot (see sections 5.3 and 5.4). In all of these experiments in location recognition, a preprogrammed and fixed wall following behaviour was used.

It seems logical to combine both sets of experiments into *one* experiment; firstly this would produce a more flexible robot controller (predefinition limits flexibility; if preprogrammed behaviour can be avoided, the robot will be better able to achieve its task in the particular environment it is placed in), secondly we would gain more insight into how a learning in stages, a learning built on previously acquired knowledge can be achieved.

In this chapter I discuss two experiments that address exactly this question: can previously gained competences be used in subsequent learning stages, and how is the whole learning process to be structured in order that the different stages do not interfere with each other? It seems immediately obvious — and the experiments have confirmed it — that one cannot leave the robot to learn everything at the same time, in other words it is not possible to have a controller consisting predominantly of adjustable (plastic) components and hope that "everything will fall into place automatically". As will become clear from the following, some fixed structure guiding the staged learning process *is* needed!

6.2 Corridor Following and Maze Learning

Before I attempted to combine the experiments on motor competence acquisition and location recognition, I conducted an experiment whose primary aim it was to find out how the learning process could be structured in order that once acquired competences could be used subsequently. I chose to build a controller that would allow Cairngorm to first learn how to follow corridors, and then use this skill to detect junctions in a maze which consisted solely of T-junctions. Such a maze is shown in figure 6.1.

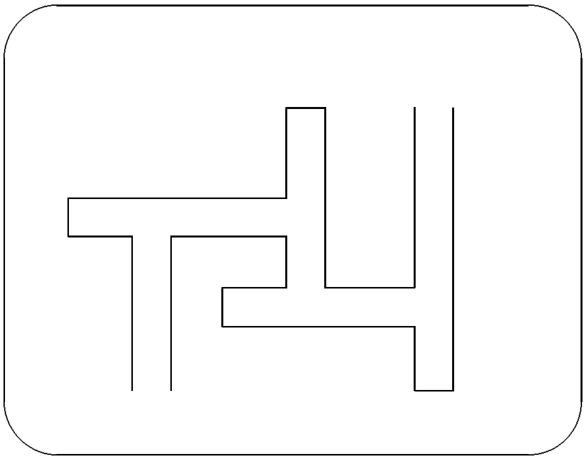

Figure 6.1: A typical maze Cairngorm is able to negotiate.

To make the experiment a little more interesting, the robot not only had to recognise junctions in the maze, but also dead ends. In addition to that, the robot was to learn which way to turn at each junction in order to move towards the exit on the shortest route. To do this, the acquired corridor following behaviour was to be used. If both junctions and dead ends could successfully be recognised, the robot could be made to learn which way to turn at each junction in order to find the shortest way to the exit: whenever a junction was detected, that was the time to take a decision; whenever a dead end was detected, the decision made earlier was wrong, whenever it was another junction the decision made earlier was correct.

In order to recognise landmarks such as junctions or dead ends, a suitable method is to consider the time a particular motor action lasts. If the robot is trying to move away from an obstacle, that is if a whisker sensor is on, and the time it takes the robot to come free exceeds either some preset threshold or it is longer than the average of obstacle avoidance actions (see chapter 5.4), with high probability the robot is in a dead end. Similarly, if the robot is trying to touch a wall, that is if the boredom instinct-rule is triggered, and the time it takes the robot to touch the wall exceeds a preset threshold or is longer than the average of wall seeking actions, the robot is most likely at a junction. The problem with this approach is, however, that the learning of the basic motor competences has to be finished before landmarks can be detected in this way. Why is this? As discussed in chapter 4, the robot learns its task-achieving motor actions through trial and error. It will try a particular motor action for a preset amount of time, but if the violated instinct-rule does not become satisfied within this period the motor action will be abandoned, and another action will be tried instead. For detecting landmarks, i.e. locations where motor actions do exceed the duration of an average motor action, this is unsuitable. Hence the need to terminate the learning of motor actions at some stage.

What would be a suitable criterion for determining when to "freeze" the acquired knowledge and terminate the learning of motor actions? I chose the following method: in order to behave intelligently the robot ought to have encountered *every* possible input signal constellation at least once, but preferably more often than that. The input vector used to train the Pattern Associator in corridor following is shown in figure 6.2.

Time t		Time t-1		
1=Lft. Wh. on 0=Lft. Wh. off	1=Rgt.Wh. on 0=Rgt.Wh. off	1=Lft. Wh. on 0=Lft. Wh. off	1=Rgt.Wh. on 0=Rgt.Wh. off	$\neg(B1 \vee B2 \vee \ldots$ $\ldots \vee B3 \vee B4)$
B1	B2	B3	B4	B5

Figure 6.2: The input vector used for corridor following.

It has four relevant bits[1], which means that the number of different possible input constellations is sixteen. In practice, however, there are less constellations likely to occur: following a corridor hardly ever will *both* whiskers be on, instead it will be either none, or the right, or the left whisker. This means, at the most, nine different input vectors. As, because of the setup of the controller (as described in chapter 4), it is not possible that both at time t and at time t-1 *no* whiskers are on, the number of possible different input vectors is eight at the most. Therefore I chose the following rule to determine when to finish the learning of motor competences: if the robot has not made a mistake at all in the choice of motor action eight times in a row, the acquired knowledge is frozen and subsequently used for landmark detection.

6.2.1 Results

The main thrust of this first experiment in staged learning was to find a method which would allow the robot to determine autonomously

[1] Bit 5 of the input vector ensures that the input vector is *always* non-zero, but it is practically never on.

when to assume that it has learned the required motor competences and when, therefore, to freeze the acquired knowledge. The experiments show that the guiding rule of eight correct motor actions in a row satisfies this requirement. Cairngorm had always acquired a correct corridor following behaviour when it decided to terminate the learning of motor actions and continue with learning correct turns at the junctions of the maze.

Having learned how to follow a corridor, Cairngorm usually detected junctions and dead ends successfully. At each junction, the robot incremented an internal counter and made a decision whether to turn left or right (the purpose of the junction counter was to allow the robot to associate particular locations with particular turns, either to the left or to the right). If subsequently another junction was detected, the decision had been correct, if a dead end was detected it had been wrong. This way the robot associated the correct turn with each junction and was usually able to move through the maze without error. Occasionally, however, Cairngorm did not detect a junction. This, of course, upset the junction count and all subsequent learning of correct turns. As the main point of this initial experiment was to find a criterion for deciding when to stop the learning of motor actions, I did not try to remedy this fact.

6.2.2 Summary

The Experiment at a Glance

Title of Experiment: Staged learning: corridor following and maze learning.

Robot used: Cairngorm.

Sensors used: Two tactile sensors, mounted at the front of the robot, and the internal clock.

Motor actions used: Left and right turn.

Network used: Two layer Perceptron with five input nodes and four output nodes for acquisition of wall following competence (see figure 4.21).

Input vector used: Five bit input vector containing the status of the two whiskers at time t and t-1 for motor competence acquisition (see figure 6.2).

Instinct-rules used: 1.(Move forward!), 2. Keep whiskers quiet!, 3. Make whiskers signal after a while!, 4. Make alternate whiskers signal!

Magic numbers used: If the robot has performed the correct corridor following actions eight consecutive times, the corridor following competence is assumed to be established. Gain $\eta = 0.3$, as before. Both values are not critical.

Results and observations: In the first phase of the experiment Cairngorm successfully acquires the ability to follow corridors. In the second phase these motor actions are used to detect junctions and dead ends. Exploring a maze, detecting such landmarks, the robot then associates each junction with its correct turn.

6.3 Wall Following and Mapbuilding

All the mechanisms required for the final experiment are now explained: how to acquire the necessary motor competences in the first place (chapter 4), how to train a self-organising feature map (SOFM) so that it can be used for location recognition (chapter 5), and how to decide when to finish learning the former and start learning the latter (previous section). All that remained, therefore, was to combine these mechanisms in one controller. The result hoped for was a robot that would be able to acquire motor competences autonomously, decide when this learning process is accomplished and then use the acquired skills to construct a representation of its world which it could later use to recognise places again it had once visited.

6.3.1 The Experiment

To learn wall following, obstacle avoidance and dead end escape behaviour, Cairngorm used the controller described in section 4.4. In the first phase of the experiment the robot was placed in region R of the enclosure (see figure 6.3).

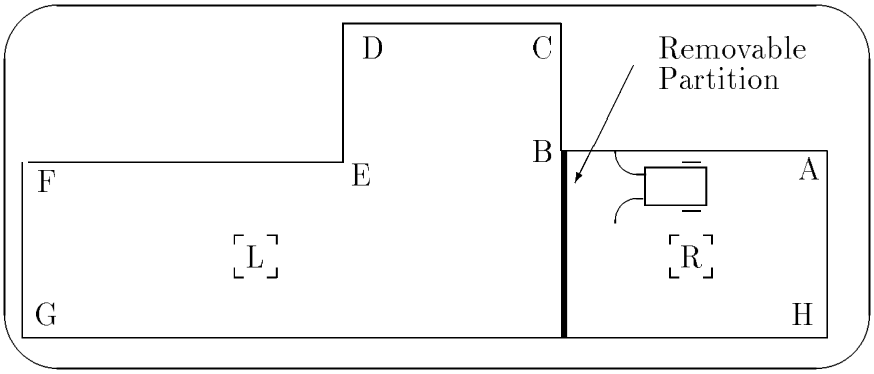

Figure 6.3: Setup for the experiment in staged learning.

The reason why region R is partitioned off and the robot placed there is that here the robot can concentrate, as it were, on learning wall following and dead end escape. Once the learning of these competences is accomplished, the robot indicates this by switching on an indicator lamp, and the partition is removed by the experimenter. As before, at this time of the experiment the learning of motor actions ceases and the acquired knowledge is frozen. One question not yet answered is how the robot decides when this point is reached. A similar consideration as in section 6.2 goes as follows: the input vector used for acquiring the wall following skill is the one shown in figure 6.2. In the context of this experiment, that is in a world that contains walls *and* dead ends, all possible input vector constellations (i.e. sixteen) do actually occur in the real world. Cairngorm's initial learning phase of motor competences is therefore ended when the robot has made eighteen (sixteen plus safety margin) correct decisions in a row. Whenever a wrong decision is made, this counter starts again from zero. After completion of the first learning phase, Cairngorm uses the acquired competence, follows the wall and develops the self-organising feature map it later uses for location recognition. In the experiment described here, the robot uses the same setup as described in section 5.4. The input vector used for developing the self-organising feature map is shown in figure 6.4; it contains similar information to input vector 2 in figure 5.16 (the only difference being that the elapsed time between significant motor actions is encoded in two bits, not three).

After about five minutes in region R the robot has usually learned to follow a wall and not to get stuck in cul-de-sacs. When the robot follows a wall on its right hand side, it learns to move right in order to touch the wall, and to turn left whenever any whisker is touched. It can happen, for rare input stimuli, that the robot turns left instead of right to find the wall, which results in a complete turn of the robot. Afterwards normal operation is resumed; this freak behaviour has no

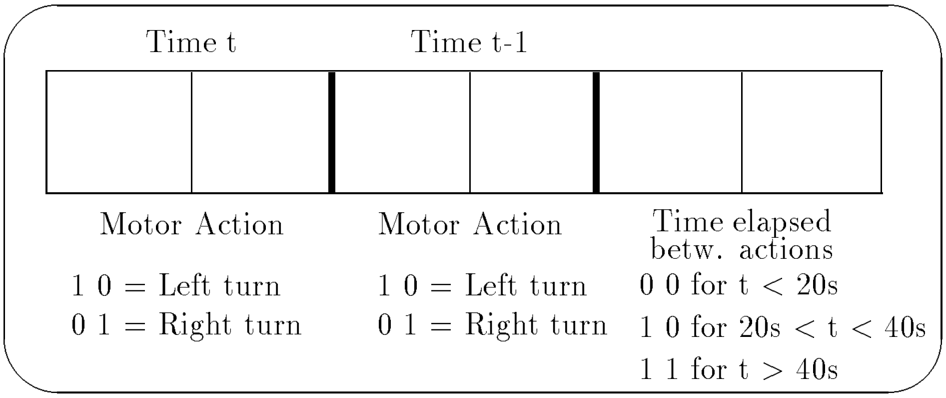

Figure 6.4: Input vector for self-organising feature map.

influence on the mapbuilding process. The acquired behaviour is very robust and in all experiments the robot followed the wall successfully.

The mapbuilding process takes about another five minutes — the time it takes Cairngorm to complete about three rounds in its enclosure. After this time the self-organising feature map is so stable that it can be used for location recognition. That the SOFM settles gradually to a stable state and allows more and more differentiation between different physical locations in the robot's world is shown strikingly by the following observation. I deliberately marked location B as the location to be recognised very early on in the self-organising process, long before the network had actually settled into a stable state. Within the same round the robot took locations E and G to be identical to B. The confusion with location E is not too surprising — both B and E are convex corners, which means that similar motor actions are performed by the robot at these locations (which must have led to the confusion), but location G is quite different. Why B and G got confused is difficult to explain; I suspect that at such an early stage almost all locations produce similar excitation patterns of the self-organising feature map (note that the distance between A and B is about the same as the distance between F and G, which means that the travel times will also be of similar magnitude). One round later, however, the confusion between B and G had disappeared (only B and E were taken to be same location), and the round after that location B was uniquely identified.

Once the network was organised, various experiments showed that, for example, locations B, G and H were recognised correctly and not confused with any other location, and that locations F and D were recognised correctly, but confused with each other. The latter observation is not surprising, because the input vectors generated at these two locations are identical (see figure 6.4). They are identical, because usually the robot does not generate an input vector at location C, which means that the time that elapses between locations B and D is about the same as the time between locations E and F.

Following page 184 there is a photograph of the experiment. Area A is the part of the enclosure where the robot acquires the wall following and dead end escape competences. At the beginning of the experiment the robot is placed at point B. The first two times the robot is looking for the wall it turns left (peaks going to the left), after that the robot turns right to touch the wall. At location C the first obstacle is in the way, a facing wall. Cairngorm (incorrectly) turns right, then left to resume its path. The distance from C to the next corner (the one under the letter A) is too short to have any wall seeking action. The obstacle avoidance on this stretch is correct (slight left curves in the photograph). Shortly after this corner the robot has forgotten how to seek the wall, it turns left instead of right (peak to the left). Eventually, the robot acquires all necessary competences, they get frozen by the robot and the partition is removed. The short peaks to the right (very clearly visible at point H) indicate (correct) wall seeking movements, curves to the left indicate an obstacle avoiding left turn of the robot. The letters correspond to the following locations in figure 6.3:

D = corner B, E = corner D, F = corner F, G = corner H.

In summary, this experiment presents a mobile robot that uses its learning and self-organising capabilites to first acquire the necessary motor competences, and then use these to explore an unknown environment and recognise locations in this environment. To achieve this, no predefined world model or predefined map is used, instead the robot uses the fixed and plastic components (instinct-rules, artificial neural networks) in its controller to build its own representation of the world it is interacting with.

Following page: Photograph of the experiment on staged learning (explanation on page 182).

6.3.2 Summary

The Experiment at a Glance

Title of Experiment: Staged learning: acquisition of wall following and obstacle avoidance competences and mapbuilding.

Robot used: Cairngorm.

Sensors used: Two tactile sensors, mounted at the front of the robot, and the internal controller clock.

Motor actions used: Left and right turn.

Network used: Two layer Perceptron with five input nodes and four output nodes for acquisition of wall following competence (see figure 4.21), and a ring shaped self-organising feature map of fifty units for location recognition (see figure 5.3).

Input vectors used: Five bit input vector containing the status of the two whiskers at time t and t-1 for motor competence acquisition (see figure 6.2), and a six bit input vector containing information about the significant motor actions of the robot (see figure 6.4).

Instinct-rules used: 1.(Move forward!), 2. Keep whiskers quiet!, 3. Make whiskers signal after a while!

Magic numbers used: If the robot has performed the correct wall following and obstacle avoidance actions eighteen consecutive times, the motor competences are assumed to be established. Gains as in previous experiments (not critical).

Results and observations: In the first phase of the experiment Cairngorm successfully acquires the ability to follow walls, move away from obstacles and get out of dead ends. In the second phase these motor actions are used to train a self-organising feature map for location recognition. As in previous experiments, Cairngorm successfully recognises locations it has visited before.

6.4 Discussion

The experiments in motor competence acquisition (chapter 4) and location recognition (chapter 5) are examples of a robot acquiring a single competence. Most work on learning in artificial intelligence has focussed on such learning of single competences. Mobile robots, however, require a number of competences in order to achieve their tasks and stay operational (like, as was the case in the experiment presented in this chapter, to avoid obstacles, to follow walls and to build maps). I have argued earlier (page 173) that the less the robot controller relies on predefined knowledge, the more flexible it will be. It is therefore desirable to let the robot acquire as many of the pre-specified competences as possible by itself, rather than used predefined knowledge. The experiments presented here are a first step towards that goal.

In the earlier experiments described in this thesis, the skills of the robots were not only single ones, inevitably they also were defined in terms of external definitions. Instinct-rules provided by the designer guided the process of motor competence acquisition, hardwired sensor-motor connections and fixed mapbuilding and location recognition procedures, again provided by the designer, controlled the experiments in location recognition. Here this is no longer the case. The competence of junction and dead end detection builds on the robot's ability to follow corridors, the location recognition competence is based on the robot's ability to follow walls, to avoid obstacles and to escape from dead ends. Later acquired competences are, to some degree, defined in terms of earlier acquired ones.

Chapter 7

Summary and Conclusion

7.1 Summary

One way of controlling a mobile robot is to construct a world model within the robot controller, using information derived from sensor data. This world model can then be used to plan the robot's actions in the world, to monitor them and to correct them ("Sense, think, act" cycle, [Malcolm *et al.* 89]). In this thesis I have shown that many of the tasks that are achieved by today's mobile robots can be implemented without any explicit world model and without planning. I have described experiments in competence acquisition, conducted with the mobile robots Alder and Cairngorm. The robots have controllers that use artificial neural networks, the competences that these robots have are acquired, not predefined. Instinct-rules guide the learning process in which the robots acquire competences such as obstacle avoidance, wall following, dead end escape and corridor following. Because these skills are *acquired*, not *fixed*, the robots can cope with unforeseen situations: swapped whiskers, changed direction of movement whilst following a wall, changed environment *et cetera* will not disable the robot, but merely confuse it for a short while until its behaviour has adapted to the new situation.

By increasing the amount of information presented to the self-organising controller of Alder and Cairngorm it was possible to improve the performance of the robots in more than one context: the robots were able to recognise particular contexts and behave accordingly. This meant that for example it was possible for them to learn to avoid obstacles *and* to escape from dead ends, without the two contexts interfering with each other.

In a further set of experiments concerning navigation self-organising feature maps were used for location recognition. Here the robots train a self-organising feature map, using sensor signals or motor actions. The excitation patterns developing on the self-organising feature map are then used for location recognition. This scheme proved to work well for both the case where input information to the self-organising feature map was derived from sensory information, and the case where it was based on the motor actions of the robot: the robots were able to extract relevant data from the input signal stream and recognise locations reliably, even in the presence of noise and variation inherent in the real world.

The final group of experiments on staged learning combined the two sets of experiments mentioned earlier. Here the robot used one acquired competence for the subsequent acquisition of another one. In the experiments Cairngorm for example first learned to follow a corridor, and then used this ability to negotiate a maze successfully. Also, in a second experiment the robot acquired the skill to avoid obstacles, escape from dead ends and follow walls, and then used this to train a self-organising feature map to recognise locations.

7.2 Open questions

Some questions concerning competence acquisition for mobile robots have been answered by the work presented in this thesis, but many also have been raised by it. Obviously, I cannot answer them in this thesis, but I will presently discuss them and make some suggestions as to how they might be tackled.

7.2.1 Scalability

Alder and Cairngorm are simple robots, normally using two tactile sensors, occasionally also odometry, ultrasonic range sensors, pushbutton switches or internal clocks. They also have a limited repertoire of motor actions: left and right turn, forward and reverse movement. How the experiments described will scale up is an important issue. In particular, I see the following five aspects of the issue of scalabilty:

1. Connecting more sensors, and of different types,
2. using robots with different dynamical properties,
3. achieving more behaviours by adding further instinct-rules,
4. recognising locations in more complex environments, and
5. taking the process of staged learning further.

I have argued on page 94 that it is an advantage of the presented controller architecture that the effective wiring between sensors and actuators is determined by the robot, autonomously. I believe it would present no particular problem if more sensors of the same kind were connected. The associative memory in the controller would grow in size, increasing the computational load, but a lot[1] of sensors could be

[1]Considering the increase in computational load alone, in Alder's case this increase can be one order of magnitude.

connected before even Alder's performance would be impaired by too big a network. The advantage, of course, of connecting many sensors of the same type is in the redundancy this brings; broken or ineffective sensors, although no longer contributing towards the eventual motor action taken by the robot, do not inhibit successful performance: other sensor signals take over. Some work has already been done on this question of portability and scalability: [Daskalakis 91] successfully implemented an instinct-rule based self-organising controller as described in chapter 4 on Lego robots ([Donnett & Smithers 91]), which have more sensors than Alder and Cairngorm, and which are also faster.

Similarly, sensors of different type could be connected, the necessary sensor fusion would be performed by the associative memory. The adequate relative weighting of different sensors in the input vector is an interesting research topic that arises in this case; weights ought to reflect sensor range, reliability and redundancy in data coming from different sensors. It is widely accepted in psychology that in humans, for example, sensors definitely have different weights (vision being the most important), and that sensors generate complementary, not redundant signals.

Increasing the behavioural repertoire of the robots by simply adding further instinct-rules is an exciting prospect, it makes the design of robot controllers much easier. The question, however, of how to determine the required new instinct-rules is not yet conclusively answered. So far I have used rules of thumb which are intuitively easy to understand, but a methodology is as yet lacking. Also, as I have argued in section 4.6, some arbitration procedure or precedence definition between instinct-rules might become necessary if more than one instinct-rule can be active at a time. An interesting, open question therefore is: how complex and how deeply nested can the robot's behaviours be, using instinct-rule based controllers as outlined in chapter 4? As a further extension to this question it would be interesting to investigate whether the determination of instinct-rules could be automated.

The mapbuilding scheme presented here has only been tried in simple environments. How it would perform in a more complicated environment (for example an environment where more than one canonical path exists)? What sort of input vectors and what sort of self-organising feature maps should be used? These are open questions. I believe that it is possible to use an architecture as described, for example, in section 5.4, even in a far more complicated world. The size, and possibly also the dimensionality of the network used will have to be increased, but in principle it should be possible.

For the first time the autonomous acquisition of multiple competences in a mobile robot has been presented. I believe that this process can be extended even further than has been done here, for example the mapbuilding process could be consolidated at some point and form the basis for a further competence, that of map interpretation.

7.2.2 Robustness

How robust are the control architectures presented here? Will the mapbuilding, for example, cope with changing environment, with moving obstacles etc.? The experiments presented here have already shown that the robots can cope with variation inherent in the world, for the acquisition of motor skills, for example, it is not necessary that the robots receive 'perfect' teaching signals in order to learn effective mappings from sensor to actuator space. However, more investigations in the robustness of these schemes, particularly the mapbuilding scheme, is required to establish more clearly the robustness of this type of scheme.

7.2.3 Navigation

As mentioned before, a true navigational skill requires both location recognition and map interpretation. The latter, of course, is not implemented in Alder and Cairngorm, which is the reason why the robots can only reach locations along a canonical path (following the wall of their enclosure).

I think there is a way of implementing a map interpretation skill in mobile robots, using ideas presented in this thesis. The mapbuilding process could be performed exactly as described in chapter 5; then, in a second stage, another connectionist computing architecture could learn to associate changes of excitation patterns perceived on the self-organising feature map with motor actions performed by the robot. This would happen in an exploratory phase, until this second map, representing the relationship between motor actions and excitation centres of the self-organising feature map, is stable. It could then be used by the robot for navigation, not only along a canonical path, but along an arbitrary path. This idea was first mentioned in [Nehmzow & Smithers 91a] (see appendix B).

Concerning location recognition and the experiments presented in this thesis, an important question is how big the self-organising feature map (SOFM) has to be, and of what dimensionality.

I have addressed this question in section 5.2.2 and given some guidelines on determining the required size and dimensionality. Obviously, it is impossible to map a higher dimensional input space onto a lower dimensional SOFM whilst preserving all the topology. Nevertheless such mappings ([Hertz *et al.* 91, p.238] give an example) might be sufficient for robot control, it is a question worth investigating.

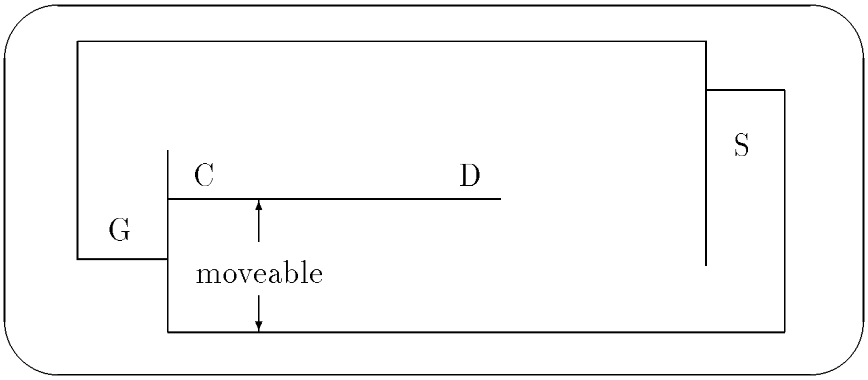

Figure 7.1: The rat intelligence test.

7.2.4 Learning Sequences

Immediate feedback regarding the success of an action is not always available; at times a sequence of actions has to be performed before success or failure are established. Figure 7.1 shows the *rat intelligence test* described by Donald Hebb, in which rats have to find the shortest way from S to G. A simple strategy by which to do this well is to follow the right hand wall. However, in terms of actual turns at corners, a sequence of actions has to be performed before the rat arrives at location G.

The instinct-rule based controller which Alder and Cairngorm use to acquire motor skills is not suitable for learning such sequences. One way this might be achieved, though, is to use self-organising feature maps. In the location recognition experiments, the robots recognise locations not by the mere properties of these places, but by the history of events that took place along the way towards that location, in other words: by the sequence of events. Similarly, sequences of actions could be mapped onto units of a self-organising network, and the excitation patterns of the network be associated with reward or punishment.

7.2.5 Adjustable Plasticity

The question of habituation is an important one. Even for the acquisition of single motor competences a consolidation of the learning process may be desirable, to obtain a more stable behaviour of the robot, possibly also to free computational resources for other tasks. To some extent habituation is a side effect of the learning process used in Alder and Cairngorm. The longer consistent input stimuli are received, the more a Perceptron settles in a stable state, and the more 'forgetting' is required before the network learns a new mapping between input and output state.

In the experiments in staged learning (chapter 6), however, Cairngorm acquires one competence, and then completely freezes the acquired knowledge to use it for further learning (this is an extreme form of habituation). That is not necessarily a good thing. Obviously, the advantage that the robot is able to adapt to changing circumstances is lost if the learning process is stopped. A more flexible approach would be if some ability for learning fundamental competences always remained, even though it would be tuned down in favour of the learning of more complex tasks. Such a learning might be modulated by a novelty detector, increasing the learning rate as soon as unusual events are detected. As the experiment in staged learning stands at the moment, Cairngorm could cope with for example swapped whiskers as long as it was still learning to follow a corridor or a wall. As soon as this learning is achieved (and therefore frozen), however, swapped whiskers would be fatal: the robot would be performing the wrong actions, without ever being able to recover.

Adjustable plasticity is also an issue in mapbuilding. A single obstacle, encountered during the mapbuilding process, should be ignored. If, however, it is encountered persistently, it should be mapped. Such behaviour might be achieved by using a time-dependent gain in the learning algorithm for the self-organising feature map: the learning rate would become less and less, so that the map would eventually settle in a stable state. Again, as suggested above, this gain could be modulated by a novelty detector.

7.3 Conclusions

People as designers of robot controllers do not have the experience of a robot, undeniably so, which makes it impossible for them to reliably determine all the features a robot controller will require *a priori*. The more dependent on designer-defined knowledge a robot controller is, the less flexible it will inevitably be when in operation. This work has made a number of contributions towards the understanding of how to build controllers that allow robots to autonomously acquire single and multiple task-achieving competences, how to become less dependent on designer-defined knowledge and how to achieve higher degrees of flexibility in a world that is noisy and subject to variation.

Little work has been done to date on actual implementations of learning algorithms on mobile robots, two examples of related work are presented in [Mahadevan & Connell 91] and [Maes & Brooks 90]. Compared to these, and compared to related simulations of reinforcement learning architectures ([Kaelbling 90, Prescott & Mayhew, Sutton 91]), the self-organising controller presented in this thesis and implemented on Alder and Cairngorm offers extremely fast learning. Motor competences such as obstacle avoidance, wall following, corridor following or dead end escape are learned in two or three learning steps[2], requiring under one minute[3] of operation of the robot. Because the effective wiring between sensors and actuators is autonomously established by the robot, not predefined by the designer, setting up of robots is made easier and less prone to error. It could furthermore be shown that the behavioural repertoire of the robots can very easily be expanded through addition of further instinct-rules; to achieve corri-

[2]More than 2000 learning steps are needed in Obelix' case to learn box finding ([Mahadevan & Connell 91]).

[3]Genghis needs ten minutes to learn to walk, if guiding heuristics are given the learning process is accelerated to 1' 45" ([Maes & Brooks 90]).

dor following, for example, an addition of a mere five lines of code to the wall following controller is required. Also, the learning achieved is so fast that it offers a viable alternative to rule-based error recovery strategies ([Penders 89, Spreng 91]), built to improve a robot's ability to cope with abnormal situations.

This thesis presents a novel approach to coping with unforeseen situations in mobile robot control, a topic that has not been addressed before. It was shown that, through the interaction of fixed and plastic components in the controller, mobile robots can cope with situations truly unforeseen by the designer, situations such as changes in the robot morphology, changes in the environment and the task. To cope with unforeseen situations is beyond the scope of rule-based systems, simply because the error situation is by definition unidentified and has never been encountered before.

A reliable and effective mapbuilding architecture is presented which can be used for location recognition. As before, this process is extremely fast. In an enclosure having eight corners as landmarks, this architecture produces usable maps within twenty-five input vector presentations which, at the slow travelling speed of Alder and Cairngorm, takes about five minutes. The process is reliable in the presence of noise and variation and computationally so cheap that it can be executed in real time using an eight-bit 8052 microprocessor, programmed through a BASIC interpreter.

For the first time the autonomous acquisition of multiple competences in mobile robots has been presented. Such staged learning is the foundation of autonomous acquisition of higher level competences, for example navigation or delivery tasks. Cairngorm learns fundamental motor skills such as wall following, and subsequently uses these for the acquisition of further competences, such as for example mapbuilding. These secondary skills are defined in terms of the primary skills —

the robot, for example, recognises junctions in a maze through its previously acquired corridor following behaviour — which means that a grounding of competences in terms of the robot's experience space, not in terms of the designer's definitions, is achieved. This is a further step away from flexibility-limiting predefinition.

The architectures used for the acquisition of motor-sensory skills as well as the ones used for location recognition (apart from the architecture described in section 5.3) are easy to implement, easy to set up and work reliably, even in the presence of noise and variation.

The experiments with Alder and Cairngorm showed not only that it is possible to let robots acquire the skills they need to stay operational, to build internal representations of their world and to recognise places — they also surprised me at times. More than once the robots behaved contrary to my expectations, two examples of such incidents are mentioned in this thesis. When Alder was programmed to turn away from a signalling whisker, in many cases it did not oscillate forever in a dead end, but escaped (page 10). When Alder was programmed to imitate its simulator, Driesh, it also escaped from a dead end, unlike Driesh (page 58). The more I worked with the robots, the more I realised how difficult it is to precisely predict what will happen in experiments with robots, interacting with the world. Today I am even more convinced than I was at the beginning of my work that actually making ("real") robots is a good way of building better robots.

This thesis has addressed the problem of competence acquisition in autonomous mobile robots. The experimental results presented here are encouraging and show that autonomous competence acquisition is a suitable way towards intelligent behaviour of mobile robots and their increased autonomy and flexibility in unforeseen situations.

Bibliography

[Albus 84] J.S. Albus, *Robotics*, in: M. Brady et al. (eds.), *Robotics and Artificial Intelligence*, pp. 65-93, Springer Verlag Berlin, Heidelberg, New York, 1984.

[Allman 77] John Allman, *Evolution of the Visual System in Early Primates*, in: Progress in Psychobiology and Physiological Psychology 7, pp.1–53, 1977.

[Arkin 89] R. Arkin, *Motor Schema Based Mobile Robot Navigation*, Intern. J. Robotics Research, August 1989.

[Åström 89] Karl Johan Åström, *Toward Intelligent Control*, IEEE Control Systems Magazine, April 1989.

[Babloyantz 91] Agnessa Babloyantz (ed.), *Self-Organization, Emerging Properties, and Learning*, Plenum Press, New York 1991.

[Barhen *et al.* 89] J. Barhen, W.B. Bress & C.C. Jorgensen, *Applications of Concurrent Neuromorphic Algorithms for Autonomous Robots*, in Eckmiller and von der Malsburg (eds.), *Neural Computers*, Springer, Berlin, Heidelberg, New York, 1989.

[Barlow 89] H.B. Barlow, *Unsupervised Learning*, Neural Computation 1, 295-311, MIT 1989.

[Barraquand & Latombe 90] Jérôme Barraquand and Jean-Claude Latombe, *A Monte-Carlo Algorithm for Path Planning with many Degrees of Freedom*, Proc. IEEE Robotics and Automation, 1990.

[Barto *et al.* 83] Andrew G. Barto, Richard S. Sutton and Charles W. Anderson, *Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems*, in: IEEE Trans. on systems, man and cybernetics, Vol. SMC-13, No. 5,pp. 834 - 846, September/October 1983.

[Barto 90] Andrew G. Barto, *Connectionist Learning for Control*, in [Miller *et al.* 90].

[Beale & Jackson 90] R. Beale and T. Jackson, *Neural Computing: An Introduction*, Adam Hilger, Bristol, Philadelphia and New York, 1990.

[Beer 90] Randall D. Beer, 1990. *Intelligence as Adaptive Behaviour*, Academic Press.

[Braitenberg 84] Valentino Braitenberg, *Vehicles*, MIT Press, Cambridge Mass. and London, England, 1984.

[Brooks 85] Rodney Brooks, *A Robust Layered Control System for a Mobile Robot*, MIT AI Memo 864, 1985.

[Brooks 86] Rodney Brooks, *Achieving Artificial Intelligence through Building Robots*, MIT AI Memo 899, May 1986.

[Brooks 87] Rodney Brooks, *Planning is just a way of avoiding figuring out what to do next*, MIT working paper 303, September 1987.

[Brooks *et al.* 88] Rodney Brooks, Jonathan Connell and Peter Ning, *Herbert: A Second Generation Mobile Robot*, MIT AI Memo 1016. January 1988.

[Brooks 90a] Rodney Brooks, *Challenges for Complete Creature Architectures*, in: Simulation of Adaptive Behaviour, MIT Press Cambridge Mass. and London, England, 1990.

[Brooks 90b] Rodney Brooks, *Elephants Don't Play Chess*, Robotics and Autonomous Systems 6, 3-15, North-Holland 1990.

[Brooks 91a] Rodney Brooks, *Artificial Life and Real Robots*, Proceedings of 1st European Conference on Artificial Life 1991, MIT Press Cambridge Mass. and London, England, 1991.

[Brooks 91b] Rodney Brooks, *Intelligence without Reason*, IJCAI 91, pp. 569-595.

[Canny 86] John F. Canny, *A Computational Approach to Edge Detection*, in: IEEE PAMI 8(6), pp. 679–698, 1986.

[Cartwright & Collett 83] B.A. Cartwright and T.S.Collett, *Landmark Learning in Bees*, in: Journal of Comparative Physiology 151, pp. 521-543, 1983.

[Churchland 86] Patricia Smith Churchland, *Neurophilosophy*, MIT Press Cambridge Mass. and London, England, 1986.

[Clark 87] Andy Clark, 1987. *Being there: Why Implementation Matters to Cognitive Science*, Artificial Intelligence Review, Vol 1, pp. 231–244.

[Clark *et al.* 88] Sharon A. Clark, Terry Allard, William M. Jenkins and Michael M. Merzenich, *Receptive Fields in the Body Surface Map in Adult Cortex Defined by Temporally Correlated Inputs*, Nature 332 (31), pp. 444–445, March 1988.

[Collett 87] T.S.Collett, *Insect Maps*, TINS, Vol. 10 No. 4, 1987.

[Connell 89] Jonathan Hudson Connell, *A Colony Architecture for an Artificial Creature*, MIT AI TR-1151, 1989.

[Daskalakis 91] Nikolas Daskalakis, *Learning Sensor-Action Coupling in Lego Robots*, MSc Thesis, Department of Artificial Intelligence, Edinburgh University, 1991.

[De Almeida & Melin 89] R. De Almeida and C. Melin, *Exploration of Unknown Environments by a Mobile Robot*, in [Kanade *et al.* 89].

[Dickmanns & Christians 89] E. Dickmanns and Th. Christians, *Relative 3D-State Estimation for Autonomous Visual Guidance of Road Vehicles*, in [Kanade *et al.* 89].

[Donnett & Smithers 91] Jim Donnett and Tim Smithers, *Lego Vehicles: A Technology for Studying Intelligent Systems*, in [SAB 91].

[Durbin & Willshaw 87] R. Durbin and D. Willshaw, *An Analogue Approach to the Travelling Salesman Problem, Using an Elastic Net Method*, Nature (326) No. 6114, pp. 689-691, 1987.

[Fikes *et al.* 72] R.E. Fikes, P.E. Hart and N.J. Nilsson, *Learning and Executing Generalized Robot Plans*, AI, 3 (1972), pp. 251-288.

[Flynn & Brooks 88] Anita M. Flynn and Rodney A. Brooks, *MIT Mobile Robots – What's next?*, in: Proc. IEEE Conf. on Robotics and Automation, 1988.

[Freund *et al.* 91] E. Freund, R. Mayr, F. Dierks, U. Judaschke, U. Kernebeck, B. Lammen, *Safety Aspects for Autonomous Robot Systems*, in [Schmidt 91].

[Fröhlich *et al.* 91] C. Fröhlich, F. Freyberger, G. Karl and G. Schmidt, *Multisensor System for an Autonomous Robot Vehicle*, in [Schmidt 91].

[Giralt *et al.* 84] G.G. Giralt, R. Chatila and M. Vaisset, *An Integrated Navigation and Motion Control System for Autonomous Multisensory Mobile Robots*, in: Robotics Research, 1st international symposium, pp. 191 - 214, 1984.

[Goto & Stentz 87] Y. Goto, A. Stentz, *The CMU System for Mobile Robot Navigation*, in: Proc. IEEE Conf. on Robotics and Automation, 1987.

[Gould 82] James L. Gould, *Ethology: The Mechanisms and Evolution of Behavior*, W.W. Norton and Co., New York 1982.

[Gould & Gould 88] James L. Gould and Carol Grant Gould, *The Honey Bee*, p. 106, Scientific American Library, New York, 1988.

[Hasler *et al.* 78] Arthur D. Hasler, A.T. Scholz and R.M. Horrall, *Olfactory Imprinting and Homing in Salmon*, American Scientist 66 (1978) pp. 347–55; quoted in [Gould 82].

[Hebb 49] D.O. Hebb, *The Organization of Behavior*, Wiley, New York 1949.

[Hertz *et al.* 91] John Hertz, Anders Krogh and Richard Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley 1991.

[Hubel 79] David H. Hubel, *The Visual Cortex of Normal and Deprived Monkeys*, American Scientist, 67 No 5, pp. 532–543, 1979.

[IEEE 91] Proc. IEEE Conference on Robotics and Automation, Sacramento 1991, pp. 674-701, 898-925.

[Kaelbling 90] Leslie Pack Kaelbling, *Learning in Embedded Systems*, Stanford Technical Report TR-90-04, June 1990.

[Kaelbling 91] Leslie Pack Kaelbling, *An Adaptable Mobile Robot*, Proceedings of 1st European Conference on Artificial Life 1991, MIT Press Cambridge Mass. and London, England, 1991.

[Kacandes *et al.* 89] Peter Kacandes, Achim Langen and Hans-Jürgen Warnecke, *A Combined Generalized Potential Fields/Dynamic Path Planning Approach to Collision Avoidance for a Mobile Autonomous Robot Operating in a Constrained Environment*, in [Kanade *et al.* 89].

[Kampmann & Schmidt 89] Peter Kampmann and Günther Schmidt, *Multilevel Motion Planning for Mobile Robots Based on a Topologically Structured World Model*, in [Kanade *et al.* 89].

[Kampmann & Schmidt 91] Peter Kampmann and Günther Schmidt, *Indoor Navigation of Mobile Robots by Use of Learned Maps*, in [Schmidt 91].

[Kanade *et al.* 89] T. Kanade, F.C.A. Groen and L.O. Hertzberger (eds.), *Intelligent Autonomous Systems 2*, Proceedings of IAS 2, ISBN 90-800410-1-7, Amsterdam 1989.

[Khatib 85] O. Khatib, *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*, IEEE Robotics and Automation 1985.

[Kohonen 82a] Teuvo Kohonen, *Clustering, Taxonomy and Topological Maps of Patterns*, Proc. 6th intern. Conf. on Pattern Recognition, October 1982.

[Kohonen 82b] Teuvo Kohonen *Self-Organized Formation of Topologically Correct Feature Maps*, Biological Cybernetics 43, pp.59-69, 1982.

[Kohonen 88] Teuvo Kohonen, *Self Organization and Associative Memory*, Springer Verlag, Berlin, Heidelberg, New York, 2nd edition, 1988.

[Knieriemen & v.Puttkamer 91] T. Knieriemen and E. von Puttkamer, *Real-Time Control in an Autonomous Mobile Robot*, in [Schmidt 91].

[Knieriemen 91] Thomas Knieriemen, *Autonome Mobile Roboter*, BI Wissenschaftsverlag, Mannheim, 1991.

[Koren & Borenstein 91] Yoram Koren and Johann Borenstein, *Potential Field Methods and their inherent Limitations for Mobile Robot Navigation*, IEEE Robotics and Automation 1991.

[Krogh & Thorpe 86] B. Krogh and C. Thorpe, *Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles*, IEEE Robotics and Automation 1986.

[Lampinen 91] Jukko Lampinen, Lappeenranta Institute of Technology, personal communication.

[Lee & Drysdale 81] D. Lee and T. Drysdale, *Generalization of Voronoi Diagrams in the Plane*, SIAM J. Comput. 10, 1981, pp. 73–87.

[Lee & Preparata 84] D. Lee and F. Preparata, *Computational Geometry — A Survey*, IEEE Trans. on Comp. 33, 1984, pp. 1071–1101.

[Levi 87] Paul Levi, *Principles of Planning and Control Concepts for Autonomous Mobile Robots*, in: Proc. IEEE Conf. on Robotics and Automation, p. 874, 1987.

[Linsker 88] R. Linsker, *Self-Organization in a Perceptual Network*, Computer, March 1988, 105-117.

[Long-Ji 91] Lin Long-Ji, *Self-Improving Reactive Agents: Case studies of Reinforcement Learning Frameworks*, in: [SAB 91].

[McGonigle & Chalmers 77] Brendan McGonigle and Magaret Chalmers, *Are Monkeys Logical*, Nature, vol 267, pp 694–696, 1977.

[McGonigle 91] Brendan McGonigle, *Incrementing Intelligent Systems by Design*, in [SAB 91].

[McGonigle & Chalmers 92] Brendan McGonigle and Magaret Chalmers, *Intelligent systems: A Cognitive Analysis*, Columbia Press, in preparation, 1992.

[Maes & Brooks 90] Pattie Maes and Rodney Brooks, *Learning to Coordinate Behaviors*, Proc. AAAI 1990.

[Mahadevan & Connell 91] Sridhar Mahadevan and Jonathan Connell, *Automatic Programming of Behavior-based Robots using Reinforcement Learning*, 9th National Conference on Artificial Intelligence, AAAI 1991.

[Malcolm & Smithers 88] Chris Malcolm and Tim Smithers, *Programming Assembly Robots in Terms of Task Achieving Behavioural Modules: First Experimental Results*, Proc. Intern. Adv. Robotics Programme, Second Workshop on Manipulators, Sensors and Steps towards Mobility, Manchester 1988.

[Malcolm *et al.* 89] Chris Malcolm, Tim Smithers and John Hallam, *An Emerging Paradigm in Robot Architecture*, in [Kanade *et al.* 89].

[Malcolm 91] Chris Malcolm, personal communication, 1991.

[Marr & Hildreth 80] David C. Marr and Ellen C. Hildreth, *Theory of Edge Detection*, in: Proc. R. Soc. London 207, pp. 187–217, 1980.

[Martin *et al.* 90] Fred Martin, Mitchel Resnick and Brian Silverman, *Braitenberg Bricks: A Lego-based Creature-Construction Kit*, Second Artificial Life Conference, February 1990, Center for nonlinear studies, Santa Fe Institute, Santa Fe, New Mexico.

[Mataric 91] Maja Mataric, *Navigating with a Rat Brain: A Neurobilogically-Inspired Model for Robot Spatial Representation*, in [SAB 91].

[Mataric 92] Maja Mataric, personal communication, June 1992.

[Matlin 89] Margaret W. Matlin, *Cognition*, Holt, Rinehart and Winston Inc., 2nd edition 1989.

[Miller *et al.* 90] W. Thomas Miller, Richard S. Sutton and Paul J. Werbos (eds.), *Neural Networks for Control*, MIT Press, Cambridge Mass. and London, England 1990.

[Minsky & Papert 88] Marvin Minsky and Seymour Papert, *Perceptrons*, MIT Press Cambridge Mass. and London, England, 1988.

[Minsky 88] Marvin Minsky, *The Society of Mind*, Heinemann, London, 1988.

[Moravec 88] Hans P. Moravec, *Sensor Fusion in Certainty Grids for Mobile Robots*, in: AI magazine, pp. 61-74, Summer 1988.

[Nehmzow *et al.* 89] Ulrich Nehmzow, John Hallam and Tim Smithers, *Really Useful Robots*, in [Kanade *et al.* 89]. See appendix 7.3.

[Nehmzow *et al.* 90] Ulrich Nehmzow, Tim Smithers & John Hallam, *Steps Towards Intelligent Robots*, DAI Research Paper No. 502, Department of Artificial Intelligence, Edinburgh, 1990, presented at Workshop on Cognition, Biology and Robotics, Gesellschaft für Mathematik und Datenverarbeitung (GMD), St. Augustin 1990. See appendix A.

[Nehmzow & Smithers 91a] Ulrich Nehmzow and Tim Smithers, *Mapbuilding using Self-Organising Networks*, in [SAB 91]. See appendix B.

[Nehmzow *et al.* 91b] Ulrich Nehmzow, Tim Smithers and John Hallam, *Location Recognition in a Mobile Robot Using Self-organising Feature Maps*, in [Schmidt 91]. See appendix C.

[Nehmzow & Smithers 91b] Ulrich Nehmzow and Tim Smithers, *Using Motor-Actions for Location Recognition*, Proceedings of 1st European Conference on Artificial Life 1991, MIT Press Cambridge Mass. and London, England, 1991. See appendix D.

[Nehmzow & Smithers 92] Ulrich Nehmzow and Tim Smithers, *Learning Multiple Competences: Some Initial Experiments*, to be presented at workshop on "Neural Networks and a new AI" as part of ECAI 92, Vienna. See appendix E.

[Nilsson 69] N.J. Nilsson, *Mobile Automation: An Application of Artificial Intelligence Techniques*, First Intern. Joint Conf. on Artificial Intelligence, Washington DC 1969, pp. 509–520.

[Penders 89] J. Penders, *Error Recovery in a Robot System*, in [Kanade *et al.* 89].

[Praßler & Milios 90] E.A. Praßler and E.E. Milios, *Parallel Distributed Robot Navigation in the Presence of Obstacles*, IEEE Symposium on Parallel and Distributed Processing, Dallas 1990.

[Prescott & Mayhew] Tony Prescott and John Mayhew, *Obstacle Avoidance through Reinforcement Learning*, to appear in J.E. Moody, S.J. Hanson and R.P. Lippman (eds.), *Advances in Neural Information Processing Systems 4*, Sam Mateo, Morgan Kaufman, no year.

[Rembold 88] Ulrich Rembold, *The Karlsruhe Autonomous Mobile Assembly Robot*, in: Proc. IEEE Conf. on Robotics and Automation, pp. 598–603, 1988.

[Rembold & Dillmann 89] Ulrich Rembold and Rüdiger Dillmann, *The Control System of the Autonomous Mobile Robot KAMRO of the University of Karlsruhe*, in [Kanade *et al.* 89].

[Ritter 88] Helge Ritter, *Selbstorganisierende neuronale Karten*, PhD Thesis, Department of Physics, Technische Universität München, 1988.

[Ritter *et al.* 89] Helge Ritter, Thomas Martinetz and Klaus Schulten, *Topology-preserving maps for Learning Visuo-motor-coordination*, Neural Networks, Vol. 2 , Pergamon Press, 1989.

[Rosenblatt 62] Frank Rosenblatt: *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan, Washington DC, 1962.

[Rumelhart & McClelland 86a] David E. Rumelhart, James L. McClelland and the PDP Research Group, *Parallel Distributed Processing*, Vol.1 "Foundations", MIT Press, Cambridge Mass. and London, England, 1986.

[Rumelhart & McClelland 86b] James L. McClelland, David E. Rumelhart and the PDP Research Group, *Parallel Distributed Processing*, Vol.2 "Psychological and Biological Models", MIT Press, Cambridge Mass. and London, England, 1986

[Rumelhart *et. al* 86c] Rumelhart, Hinton and Williams, *Learning Internal Representations by Error Propagation*, in: Rumelhart and McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge Mass. and London, England, 1986.

[Rumelhart & McClelland 86d] Rumelhart and McClelland, *Competitive Learning*, in: Rumelhart and McClelland, *Parallel Distributed Processing*, MIT Press, Cambridge Mass. and London, England, 1986.

[SAB 91] Jean-Arcady Meyer and Stewart Wilson (eds.), *From Animals to Animats*, Proc. 1st Intern. Conf. on Simulation of Adaptive Behaviour, MIT Press, Cambridge Mass. and London, England, 1991.

[Schmidt 91] G. Schmidt (ed.), *Information Processing in Autonomous Mobile Robots*, Springer Verlag, Berlin, Heidelberg, New York, 1991.

[Sherry & Schacter 87] D.F. Sherry and D.L. Schacter, 1987. *The Development of Multiple Memory Systems*, Psychological Review, vol 94, pp 439–454.

[Shewchuk & Viola 90] John Shewchuk and Paul Viola, *Implementing a Learning System for Subsumption Architectures*, IBM, T.J. Watson Laboratory, Yorktown Heights, New York, May 7th 1990.

[Simon 69] Herbert Simon, *The Sciences of the Artificial*, MIT Press, Cambridge MA 1969.

[Simon *et al.* 90] David A. Simon, Lee E. Weiss and Arthur C. Anderson, *Self-Tuning of Robot Program Primitives*, Proc. IEEE Robotics and Automation 1990.

[Skewis *et al.* 91] T. Skewis, J. Evans, V. Lumelsky, B. Krishnamurthy, B. Barrow, *Motion Planning for a Hospital Transport Robot*, in: Proc. IEEE Conference on Robotics and Automation, Sacramento 1991.

[Smithers & Malcolm 87] Tim Smithers and Chris Malcolm, *A Behavioural Approach to Robot Path Planning and Off-Line Programming*, DAI Research Paper 306, University of Edinburgh 1987, published in Journal of Structural Learning, Vol. 10, pp. 137–156, 1989.

[Spreng 91] Michael Spreng, *Dealing with Unexpected Situations during the Execution of Robot Motions*, in: Proc. IEEE Conf. on Robotics and Automation, 1991.

[Steels 88] Luc Steels, *Steps towards Common Sense*, in: Proceedings of ECAI, 1988.

[Steels 89] Luc Steels, *Cooperation between Distributed Agents through Self-Organisation*, Journal on Robotics and Autonomous Systems, 1989.

[Steels 91] Luc Steels, *Towards a Theory of Emergent Functionality*, in [SAB 91].

[Sutton 91] Richard S. Sutton, *Reinforcement Learning Architectures for Animats*, in [SAB 91].

[Thatcher *et al.* 87] R.W. Thatcher, R.A. Walker and S. Giudice, *Human Cerebral Hemispheres Develop at Different Rates and Ages*, Science 256, pp. 1110–1113, 1987.

[Torras 91] Carme Torras i Genís, *Neural Learning Algorithms and their Applications in Robotics*, in [Babloyantz 91].

[Tsai *et al* 90] Jeffrey J.-P. Tsai, Mark Metea and John Cesarone, *A Knowledge-based Navigation Scheme for Autonomous Land Vehicles*, in: Applied Artificial Intelligence 4, pp. 1–14, 1990.

[Tsubouchi and Yuta 87] T. Tsubouchi and S. Yuta, *Map assisted Vision System of Mobile Robots for Reckoning in a Building Environment*, in: Proc. IEEE intern. Conference on Robotics and Automation, p.1978, 1987.

[Tyrrell & Mayhew 91] Toby Tyrrell and John Mayhew, *Computer Simulation of an Animal Environment*, in [SAB 91].

[Walter 50] W. Grey Walter, *An Imitation of Life*, Scientific American 182(5), 42-45, 1950.

[Walter 51] W. Grey Walter, *A Machine that Learns*, Scientific American 51, 60-63, 1951.

[Waterman 1989] Talbot H. Waterman, *Animal Navigation*, Scientific American Library, New York 1989, p.183.

[Watkins 89] Christopher J.C.H. Watkins, *Learning from Delayed Rewards*, PhD thesis, King's College, Cambridge, 1989.

[Webster 81] *Webster's Third New International Dictionary*, Encyclopaedia Britannica Inc., Chicago 1981.

[Wehner 76] Rüdiger Wehner, *Polarized-Light Navigation by Insects*, Scientific American 235 No. 1 (1976), pp.106–115.

[Whitehead & Ballard 90] Steven Whitehead and Dana Ballard, *Active Perception and Reinforcement Learning*, Neural Computation 2, pp.409-419, 1990.

[Willshaw & v.d.Malsburg 76] David Willshaw and Christoph von der Malsburg, *How Patterned Neural Connections can be set up by Self-organization*, in: Proc. R. Soc. Lond. B, 194, pp. 431–445, 1976.

The following appendix contains earlier publications concerning the work reported in this thesis. The presentation of these papers has been adjusted to the format used in the main part of the thesis, but the text has remained completely unaltered (in one case I point out a spelling mistake in the original paper). Some diagrams appear slightly different to the diagrams used in the original papers, because they became available in machine-readable format later. The information contained in the diagrams remained the same. The photographs referred to in some of the papers are identical to the photographs shown in the main part of the thesis, therefore I have not included them in the appendix, but point to the appropriate photograph in the main section.

All bibliographic references refer to the bibliography beginning on page 199.

Appendix A

Really Useful Robots[1]

U. Nehmzow, J. Hallam, T. Smithers

Department of Artificial Intelligence
Edinburgh University

[1]Published in: T. Kanade, F.C.A. Groen and L.O. Hertzberger (eds.), *Intelligent Autonomous Systems*, Proceedings of IAS 2, ISBN 90-800410-1-7, Amsterdam 1989.

A.1 Abstract

We propose a self-organizing type of controller for a mobile robot. Rather than decompose the control task into subtasks and implement these (an analytic approach) we propose to use *competence generators* which generate, through a process of self-organization, the necessary competences (a synthetic approach). We have built a mobile robot and equipped it with a first version of such a self-organizing controller.

A.2 Introduction

Controlling a mobile robot can be done in several ways: the *classical* approach is to equip the robot with control circuits, using feedback loops and traditional control mechanisms to achieve stable behaviour. This is an analytic approach: the control task is decomposed by the designer into subtasks which are then implemented, using standard control techniques.

Alternatively, there is the synthetic approach: generate the desired behaviour by combining basic competences, but using a 'bottom up' approach rather than a 'top down' one. In some sense Rodney Brooks [[Brooks 86], [Brooks 85]] at MIT has chosen this approach by constructing robot controllers from simple extended finite state machines. The synthetic step here has taken place in the designer's mind.

We propose to take this approach one step further: is it possible to implement the synthesis step within the controller itself? We propose to use *competence generators* (which are driven by what we call instincts) that synthesize the competences necessary for the desired robot behaviour. This leads to the crucial questions 'What is the relationship between global behaviour and local action?' and 'Which combinations of instincts and generated competences instincts[2] produce which local actions?'.

In a spatial sense Luc Steels has worked on the first question. His *Reaction Diffusion Dynamics* [[Steels 88]] show intelligent overall behaviour (determining the shortest path between two points on a map, regardless of obstacles), although only local actions are defined (the physics of diffusion). However, for a robot controller, definition of spatial relationships alone is not sufficient: time and causality have to be considered as well.

A.3 Self-organising Systems

One promising approach for combining instincts and plasticity to generate competence is that of self-organizing systems. These have been proposed by various workers (e.g. Willshaw and von der Malsburg

[2]The word "instincts" is a misprint in the original paper and should deleted.

[[Willshaw & v.d.Malsburg 76],[Durbin & Willshaw 87]] to account for the autogenesis of topographic mappings in the nervous system). The system we have used shows properties of both Kohonen's self-organizing net [[Kohonen 88]] and Rumelhart, Hinton and Williams' back propagation network [[Rumelhart *et. al* 86c]]. Therefore we will describe these systems first.

Kohonen's [[Kohonen 88]] self-organizing system consists of a two-dimensional array of cells which all receive the same input, the input vector $\vec{\imath}$. Each cell has an individual weight vector $\vec{w}$, the output o of each cell l is determined by the scalar product of input vector $\vec{\imath}$ and weight vector $\vec{w}$:

$$o_l = \sum_{j=1}^{n} w_{lj} \imath_j$$

where n is the number of input lines.

Self-organization takes place in the following way: after a stimulus is presented to the net, one cell is bound to respond most strongly to the particular stimulus, due to the fact that the initial weight vectors are randomly chosen. (All weight vectors are <u>unit</u> vectors, so the cell responding most strongly has a weight vector which is closest to the input vector.) The weights of this cell and of the cells in its neighbourhood are updated according to the following formula:

$$w_l(t_{k+1}) = w_l(t_k) + \eta(t_k)[\imath(t_k) - w_l(t_k)]$$

(η is the 'gain', initial value typically set at about 0.5). Weight vectors of cells outside that specified neighbourhood remain unchanged. After a number of input signal presentations and updates the net will show locally distant responses for different stimuli, i.e. a mapping has taken place.

This self-organizing system is used, for example, in Kohonen's 'Magic TV' (figure 1). The input image is a light spot which is detected by a simple camera. This camera consists of a round photocathode which is divided into three equal parts. Depending on the position of the light spot on the input plane, the three parts of the photocathode will receive different proportions of light and therefore emit different electrical signals.These signals are transferred to the self-organizing system as input signals. A number of different input signals are generated, the electrical signals are presented to the net, and the interesting observation is that after a while the position of the light spot on the input plane is reproduced on the net, i.e. cells that are in a similar position as the input light spot are most excited!

We define a <u>linear</u> system as a system where the output is a linear function of the input, similarly a <u>non-linear</u> system where the output is a non-linear function of the input. In this sense Kohonen's system is a linear system. Non-linear systems might also be interesting for our purposes (see section 4 of this paper).

The classical non-linear neural net, as described for example by Rumelhart, Hinton and Williams [[Rumelhart *et. al* 86c]] (see figure 2), consists

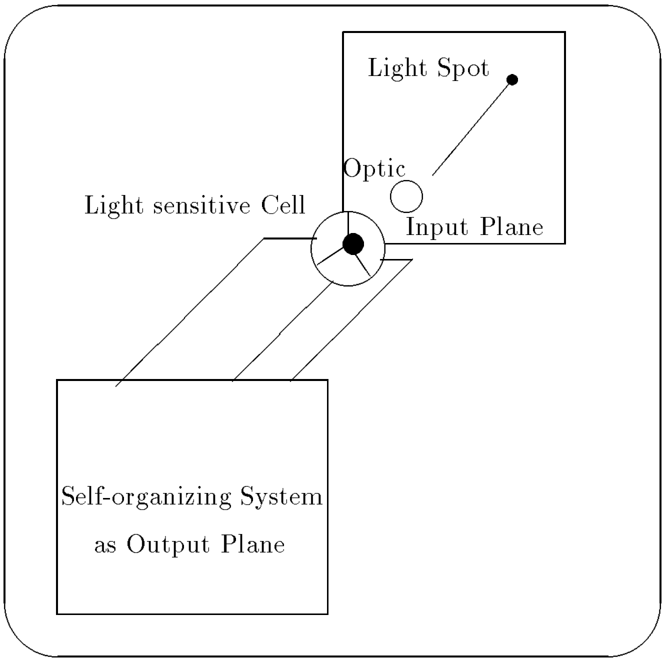

Figure A.1: The 'Magic TV' (after Kohonen)

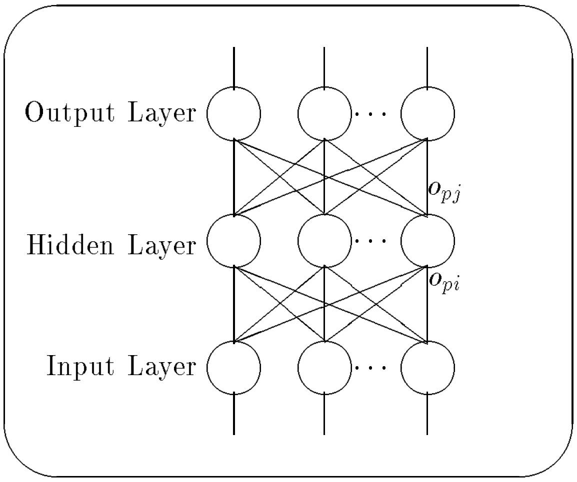

Figure A.2: Non-linear Neural Net

of a layer of input units, a layer of hidden units and one of output units. Input units simply pass the received signals on, the output of the hidden and the output units is determined by the following non-linear formula:

$$o_{pj} = \frac{1}{1 + e^{-(\Theta_j + \sum_i o_{pi} w_{ji})}}$$

where Θ is a threshold value.

Such a net is not self-organizing, but capable of 'learning'. If the desired output vector $\vec{t}$ for a given input vector $\vec{\imath}$ is known, then learning can be achieved using back-propagation. The error signal for any output unit is determined by

$$\delta_{pj} = o_{pj}(1 - o_{pj})(t_{pj} - o_{pj}).$$

For hidden units the error signal is given by

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_{k=1}^{n} \delta_{pk} w_{kj}$$

where n is the number of output units.

Updating of weights is done according to the following formula:

$$\Delta w_{ji} = \eta \delta_{pj} o_{pi}$$

$$w_{ji}(t_{k+1}) = w_{ji}(t_k) + \Delta w_{ji}$$

(again η is the gain, o_{pi} the input to cell $\imath$.) Note that here the weight vectors are <u>not unit vectors</u>.

For our mobile robot we have used a linear system which shows properties of both approaches mentioned above. Our net consists of a number of input units (sensor signals) which are directly connected to a layer of output units (motor signals), see figure 3.

Again, input units simply pass the received signals on, the output o of the output units is determined by

$$o_j = \sum_i w_{ji} i_i$$

where $\vec{w}$ is the individual weight vector of each output unit. Note that the weight vectors $\vec{w}$ are <u>unit</u> vectors, as in Kohonen's system. If the desired output vector $\vec{t}$ to a given input vector $\vec{\imath}$ is known (this vector is determined by the robot itself, using a generate-and-test method), updating the weight vectors is done according to the following rule:

$$\Delta w_{ji} = \eta (t_j - o_j) i_\imath$$

$$w_{ji}(t_{k+1}) = w_{ji}(t_k) + \Delta w_{ji}$$

and $\vec{w}$ is renormalized after this operation.

This formula differs from the one used in Kohonen's self-organizing system in the way the input vector $\vec{\imath}$ is taken into account. The expression $\eta(t_j - o_j)i_i$ is proportional to $\frac{\partial E}{\partial w_{ji}}$ with $E = \sum_j (t_j - o_j)^2$.

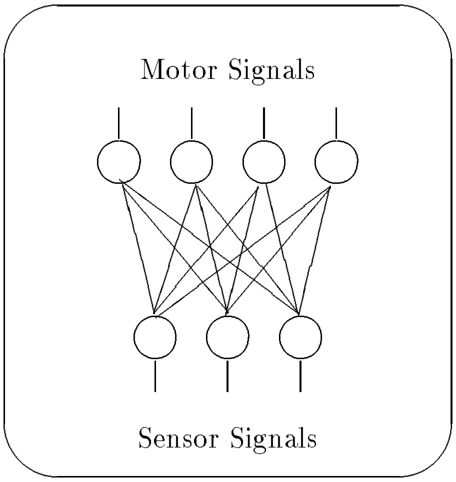

Figure A.3: Linear Net as used in our Self-Organizing Controller

A.4 Experimental Platform

Real-world situations can never be described completely by models. The real world often exhibits highly complex behaviour, therefore there will always be assumptions and simplifications that have to be made. For this reason simulation alone will not suffice. We are interested in robots interacting with the real world. Therefore experiments are very important. Our aim is to achieve true flexibility. The robot ought to be truly autonomous, independent from external knowledge as much as possible. The external knowledge we supply is reduced to some instincts which generate the basic behavioural patterns of the robot. We do not describe the task or the desired behaviour of the robot explicitly. The more general the task description is, the more information the robot has to find by itself, and the more independent it becomes.

As we said earlier: we want to look at a real robot, interacting with the real world, and being as independent from external knowledge as possible. We have therefore built such a robot, equipped with a self-organizing controller and some basic 'instincts', and watched it moving around in the real world. Hopefully our experiments will give us answers to the questions 'How is flexibility in unknown situations achieved' and 'What information is actually important for the robot, from the robot's point of view?'.

A.4.1 Preliminary Experiments

The mobile robot we use for our experiments is approximately 30 centimetres long, 25 centimetres high and 15 centimetres wide. Based on a Fischertechnik kit, the robot is equipped with two independent motors that

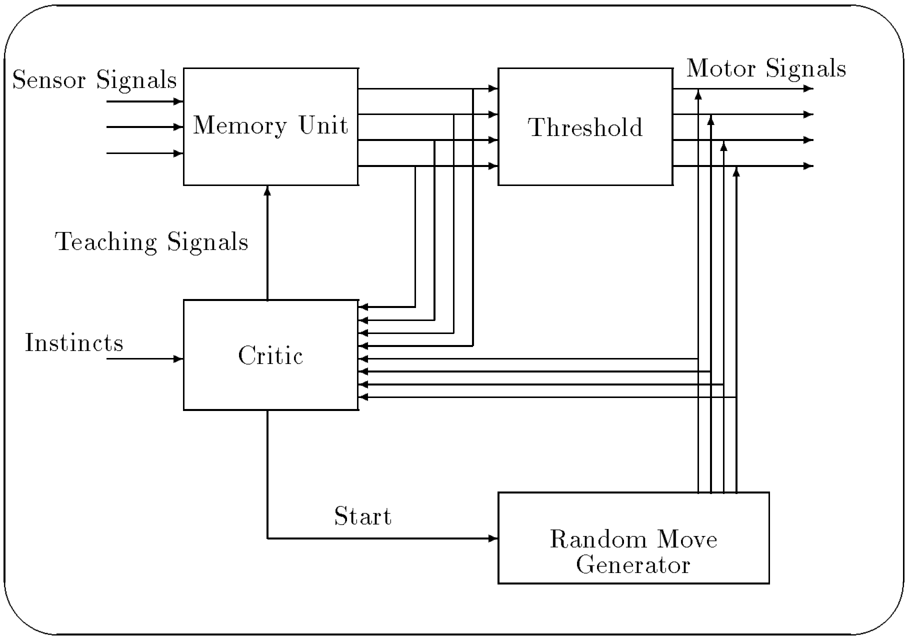

Figure A.4: Computational Structure of the self-organizing Controller

can be used for driving and steering. The driving wheels are located almost in the middle of the robot so that it can pivot round its own axis. The self-organizing control algorithm is implemented using an ARC50 computer board. This is equipped with an INTEL 8052 CPU, 32k RAM and an EPROM Basic Interpreter. We have eight input and eight outputlines connected to a specially designed interface. This I/O Interface reads the sensory signals and switches the motors on or off, according to the signals from the computer board. Currently we are using tactile sensors only (whiskers): however, any sort of binary or even analog sensor could be used. Examples of this are infrared sensors or mercury switches. The power is supplied by two lead acid rechargeable batteries, giving us a working time of approximately sixty minutes. These batteries are heavy, therefore the vehicle can move only slowly (approximately $4cms^{-1}$).

A.4.2 The Computational Structure

To control this mobile robot we use a self-organizing controller. It consists of a memory unit and a critic (see figure 4). The overall behaviour of the robot is influenced by two instincts:

1. Do not touch anything, and

2. Move forward, whenever possible.

These instincts are part of the *critic* which decides whether an action was successful (robot moved away from obstacle) or not (robot is still in contact with obstacle).

As long as no sensory signals are received ('no obstacle ahead') the second instinct will make the robot move forward. As soon as any of the whiskers detects an obstacle, however, this is no longer possible. The robot will therefore try the action that is stored in its memory unit for that particular constellation of input signals. This action will be performed for a while, typically about five seconds. If the action was successful (i.e. the robot has moved away from the obstacle) all is well and a forward move is again possible. If, however, the input signals still indicate an obstacle (and this might well be an obstacle other than the original one - the robot might have moved into more trouble) something else has to be tried. At this point a generate-and-test procedure is used, similar to the 'associative search' of Barto and Sutton [[Barto *et al.* 83]]. A random move out of either the whole set of possible motor actions or a restricted set of motor actions is selected. This move is then performed for a period of time. After each unsuccessful attempt to move away from the obstacle this period of time is extended because we assume that unless the robot moves clear of the obstacle it might have moved into even deeper problems. If, by following this method, a successfull move has been found, this move is stored in the memory unit. The next time the same set of input signals is detected this newly learned move will be performed, and hopefully be successfull straightaway. Teaching the memory unit is another task of the critic. (We have used the following teaching method in our experiments: if a move that was recalled from memory was successful, that particular move is taught once again, to confirm it; if a newly found move is to be learned, it is taught three times. The version of the net we have used has three input units (for three sensors) and four output units. The gain η is 0.2 .)

Obviously with two motors that can move either forward, backward or be switched off there are nine possible robot actions. The chances of selecting a 'good' move out of this set are smaller than picking a 'good' move from a restricted set. Such a restricted set could comprise the following actions: forward, backward, swift left and swift right. However, by restricting the set of possible actions flexibility may be reduced as well, because moves are ruled out that might be useful under certain circumstances. So far we have used the full set of robot actions.

A.4.3 Experimental Results

When switched on the robot starts moving forward, as long as there is no obstacle in the way. It is thus obeying the second built-in instinct. As soon as an obstacle is detected, the experience of the robot comes in: the robot tries the learnt move. In the beginning this move is 'do nothing' (because the memory is empty), and consequently the robot sits still (which does not solve the problem). Because the obstacle is still touched, the robot now tries another, randomly chosen action out of the set of possible actions. If this action proves to be successful, the robot will store it in its internal memory, otherwise yet another action will be tried. We observed that if the robot is left for some time, trying out moves and learning the good ones, the robot eventually behaves very efficiently. Without hesitation it turns away from an obstacle, usually using the action that will achieve this the quickest way.

There are a few very interesting side observations. For example: it doesn't matter at all which way you plug in the sensors, or the motors (as long as sensors can be read and motors switched on or off, of course). Because the system is self-organizing it learns what to do, regardless of whether a motor is plugged in 'a particular way round' or not. The same applies to the sensors: whether a sensor is read on the most significant bit, or the least, or whether it is an infrared sensor or a whisker, does not matter. And it does not matter how many sensors are actually plugged in. The controller will make use of all the information available. Furthermore we observed that if a learnt reaction was no longer successful, for example because one of the motors had failed or because the polarity of the wires had been reversed, the control algorithm would find new solutions to comply with the built-in instincts.

A.5 Discussion

Networks - regardless of being linear or non-linear - have the advantage that the problem of increased complexity can easily be handled. Adding more sensors or output functions to a self-organizing controller simply means increasing the size of the net, but not changing the general structure of the controller. In the case of our robot, for example, it means that additional sensors are simply plugged into the net, the additional information is automatically taken into account by the system.

Although we have so far used a linear net, we intend to use a non-linear net as described above in the future. Non-linear nets have a higher signal-to-noise ratio than linear nets, which increases their reliability greatly. Due to this we can make use of another propertiy of nets: in networks (both linear and non-linear) associations between input signals and reactions emerge, even if they have not been explicitly taught. Experiments show that these emergent associations are in some significant relation to their input signals. In other words: some sort of learning without teaching takes place. The high signal-to-noise ratio of non-linear nets is beneficial for the interpretation of such a net output.

A.6 Conclusion

We designed a self-organizing controller for a mobile robot. *Competence generators*, governed by instincts, generate the necessary competences (a synthetic rather than an analytic approach). We have built a mobile robot which serves as a first experimental platform. The controller of this robot is equipped with a linear net as a memory unit and a critic which provides teaching signals for the memory.

Appendix B

Steps towards Intelligent Robots[1]

DAI Research Paper No. 502

Ulrich Nehmzow, Tim Smithers and John Hallam

Department of Artificial Intelligence
Edinburgh University

(ulrich@uk.ac.ed.edai, tim@uk.ac.ed.edai, john@uk.ac.ed.edai)

[1]Presented at Workshop on Cognition, Biology and Robotics, Gesellschaft für Mathematik und Datenverarbeitung (GMD), St. Augustin 1990

B.1 Abstract

"Really Useful Robots", a research project in the domain of mobile robots, investigates new ways of controlling robots in order to achieve flexibility in unforeseen situations. This report describes the underlying ideas and mechanisms of that part of this research that has to do with competence generation.

We propose a self-organizing type of controller for a mobile robot. Rather than decompose the control task into subtasks and implement these (an analytic approach) we propose to use *competence generators* which generate the necessary competences, using *instincts* (a synthetic approach). We have built a mobile robot and equipped it with a first version of such a self-organizing controller.

B.2 Introduction

Controlling a mobile robot can be done in several ways: the *classical* approach is to equip the robot with control circuits, using feedback loops and traditional control mechanisms to achieve stable behaviour. This is an *analytic* approach: the control task is decomposed by the designer into subtasks which are then implemented, using standard control techniques.

Alternatively, there is the *synthetic* approach: generate the desired behaviour by combining basic competences, but using a 'bottom up' approach rather than a 'top down' one. In some sense Rodney Brooks [[Brooks 86], [Brooks 85]] at MIT has chosen this approach by constructing robot controllers from simple extended finite state machines. The synthetic step here has taken place in the designer's mind. We propose to take this approach one step further: is it possible to implement the synthesis step within the controller itself? We propose to use *competence generators* (which are driven by what we call instincts) that synthesize the competences necessary for the desired robot behaviour. This leads to the crucial questions 'What is the relationship between global behaviour and local action?' and 'Which combinations of instincts and generated competences produce which local actions?'. In a spatial sense Luc Steels has worked on the first question. His *Reaction Diffusion Dynamics* [[Steels 88]] show intelligent overall behaviour (determining the shortest path between two points on a map, regardless of obstacles), although only local actions are defined (the physics of diffusion).

B.3 Self-organising Systems

One promising approach for combining instincts and plasticity to generate competence is that of self-organizing systems. These have been proposed by various workers (e.g. Willshaw and von der Malsburg [[Willshaw & v.d.Malsburg 76],[Durbin & Willshaw 87]] to account for the autogenesis of topographic mappings in the nervous system, and Kohonen [[Kohonen 88]]).

Such a self-organizing system consists, for example, of a two-dimensional array of cells which all receive the same input, the input vector $\vec{\imath}$ (see figure A.1).

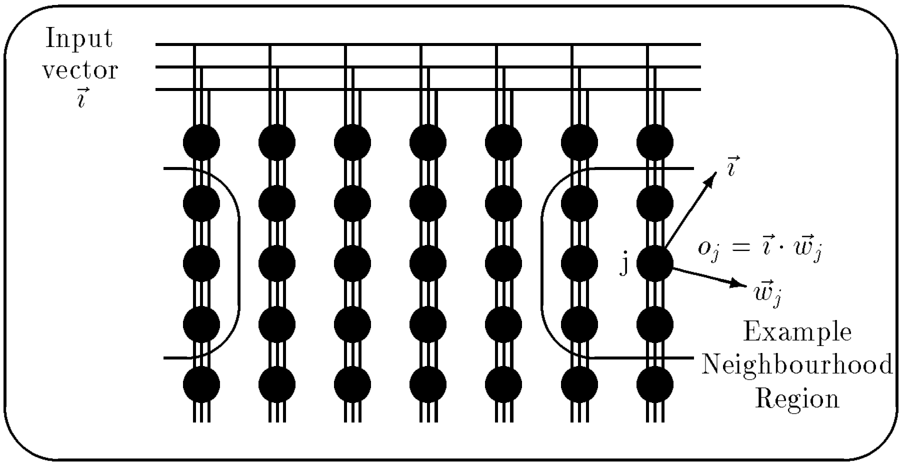

Figure B.1: Self-Organizing Network

Each cell j has an individual weight vector $\vec{w}_j$, the output o_j of each cell j is determined by the scalar product of input vector $\vec{\imath}$ and weight vector $\vec{w}_j$:

$$o_j = \vec{w}_j \vec{\imath}$$

with both $\vec{w}_j$ and $\vec{\imath}$ being normalized.

Self-organization takes place in the following way: after a stimulus is presented to the net, one cell is bound to respond most strongly to the particular stimulus, due to the fact that the initial weight vectors are randomly chosen. (All weight vectors are <u>unit</u> vectors, so the cell responding most strongly has a weight vector which is closest[2] to the input vector.) The weights of this cell and of the cells in a defined neighbourhood (which may decrease over time) are updated according to the following equation:

$$\vec{w}_j(t_{k+1}) = \vec{w}_j(t_k) + \eta(t_k)[\vec{\imath}(t_k) - \vec{w}_j(t_k)]$$

Weight vectors $\vec{w}$ are renormalized after this change. (η is the 'gain', initial value typically set at about 0.3). Weight vectors of cells outside that specified neighbourhood remain unchanged. After a number of input signal

[2]By euclidean distance.

presentations and updates the net will show locally distant responses for different stimuli, i.e. a mapping has taken place. Such a self-organizing structure performs a clustering of different input vectors, a statistical analysis.

Self-organizing structures perform *unsupervised learning.* If, however, the ability to store data is needed ('memory'), *supervised learning* schemes are needed.

The error backpropagation network [[Rumelhart *et. al* 86c]] (see figure A.2) is an example of this class of artificial neural networks.

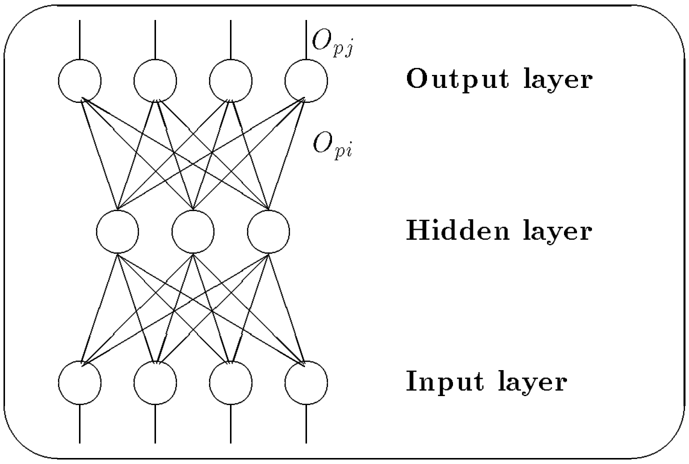

Figure B.2: Error back-propagation network

It consists of a layer of input units, a layer of hidden units and one of output units. Input units simply pass the received signals on, the output of unit j is determined by the following non-linear equation (this equation applies to both hidden and output units):

$$o_{pj} = \frac{1}{1 + e^{-(\Theta_j + \sum_i o_{pi} w_{ji})}}$$

where Θ is a threshold value and $\sum_i o_{pi} w_{ji}$ is the overall input to a unit.

Such a net is not self-organizing, but capable of 'learning'. If the desired output vector $\vec{t}$ (the "target" vector) for a given input vector $\vec{\imath}$ is known, then learning can be achieved using back-propagation. The error signal δ_{pj} for any output unit j is determined by

$$\delta_{pj} = o_{pj}(1 - o_{pj})(t_{pj} - o_{pj}).$$

For hidden units the error signal δ_{pj} is given by

$$\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_{k=1}^{n} \delta_{pk} w_{kj}$$

where n is the number of output units.

Updating of weights is done according to the following equation:

$$\Delta w_{ji} = \eta \delta_{pj} o_{pi}$$

$$w_{ji}(t_{k+1}) = w_{ji}(t_k) + \Delta w_{ji}$$

(again η is the gain, o_{pi} the input to cell $\imath$.) Note that here the weight vectors are not unit vectors.

For our mobile robot we have used a Perceptron-like [[Rosenblatt 62], [Minsky & Papert 88]] system, a so-called "pattern-associator" (see figure A.3).

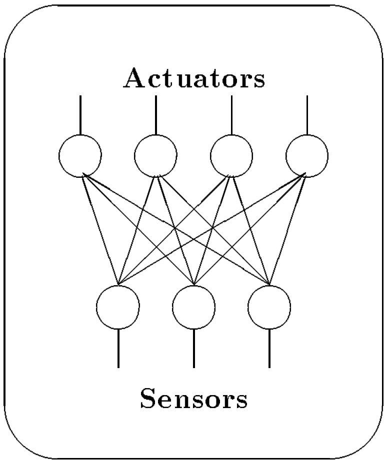

Figure B.3: Linear Net as used in our Self-Organizing Controller

The net consists of a number of input units which are directly connected to a layer of output units. The inputs contain the sensor readings at time t (and, for some applications, also the previous sensor readings), the outputs denote motor actions.

Again, input units simply pass the received signals on, the output o_j of output unit j is determined by

$$o_j = \vec{w_j}\vec{\imath}$$

where $\vec{w_j}$ is the individual weight vector of output unit j. Note that the weight vectors $\vec{w}$ are unit vectors. If the desired output vector $\vec{t}$ to a given input vector $\vec{\imath}$ is known (this vector $\vec{t}$ is determined by the robot itself, using a generate-and-test method), updating the weight vectors is done according to the following rule:

$$\Delta\vec{w_j} = \eta(t_j - o_j)\vec{\imath}$$

$$\vec{w_j}(t_{k+1}) = \vec{w_j}(t_k) + \Delta\vec{w_j}$$

and $\vec{w}$ is renormalized after this operation.

(The expression $\eta(t_j - o_j)i_i$ is proportional to $\frac{\partial E}{\partial w_{ji}}$ with $E = \sum_j (t_j - o_j)^2$.)

B.4 Experimental Platform

Real-world situations can never be described completely by models. The real world often exhibits highly complex behaviour, therefore there will always be assumptions and simplifications that have to be made if simulation is to be used. Furthermore, we are particularly interested in *flexibility in unforeseen situations*. Obviously, unforeseen situations cannot be simulated. Therefore experiments are crucial. Our aim is to achieve true flexibility. The robot ought to be truly autonomous, independent from external knowledge as much as possible. The external knowledge we supply is reduced to some instincts which generate the basic behavioural patterns of the robot. We do not describe the task or the desired behaviour of the robot explicitly. The more general the task description is, the more information the robot has to find by itself and the more independent it becomes.

As we said earlier: we want to look at a real robot, interacting with the real world whilst being as independent from external knowledge as possible. We have therefore built such a robot called 'Alder'(see page 74), equipped with a self-organizing controller and some basic 'instincts', and watched it moving around in the real world.

Hopefully our experiments will give us answers to the questions 'How is flexibility in unknown situations achieved' and 'What information is actually important for the robot, from the robot's point of view?'.

B.4.1 Preliminary Experiments

Alder is approximately 30 centimetres long, 25 centimetres high and 15 centimetres wide. Based on a Fischertechnik kit, the robot is equipped with two independent motors that can be used for driving and steering. The self-organizing control algorithm is implemented using an ARC52 computer board. This is equipped with an INTEL 8052 CPU, 16k RAM and an EPROM Basic Interpreter. The eight input and eight output lines available on the ARC52 are connected to a specially designed interface. This I/O Interface reads the sensory signals and switches the motors on or off, according to the signals from the computer board. The eight input lines can be connected to any sort of binary sensor, at the moment the robot is equipped with three tactile sensors, a revolution counter to obtain crude odometry and an "attention" button which can be pressed by the operator. In addition to these binary sensors the robot has an ultrasonic range finder, mounted either fixed in front of the base or on a stepper motor. The active range of this sonar is 25 cm to over 300 cm. The power is supplied by two lead acid rechargeable batteries, giving a working time of approximately sixty minutes. These batteries are heavy, therefore the vehicle can move only slowly (approximately $4cms^{-1}$).

B.4.2 The Computational Structure

To control this mobile robot, we use a self-organizing controller. It consists of an associative memory and a monitor (see figure A.4).

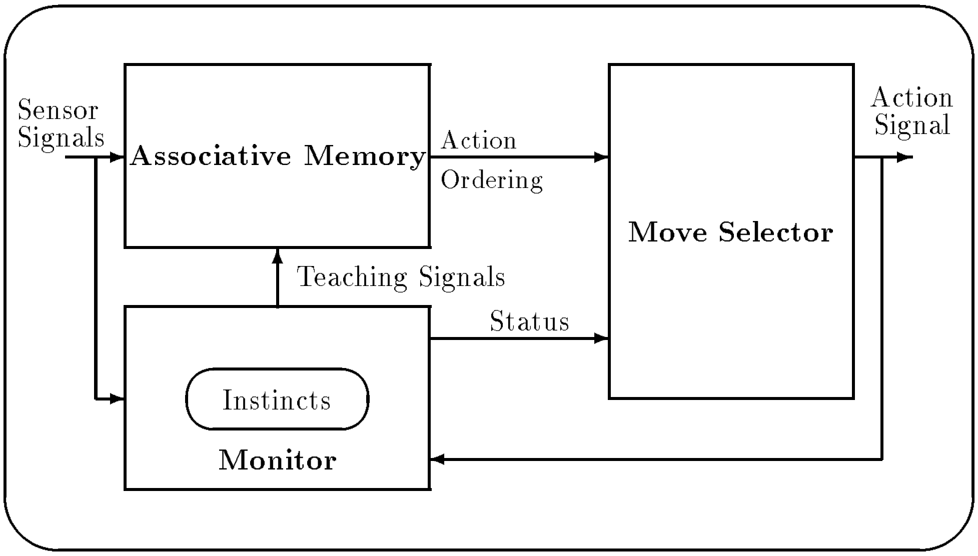

Figure B.4: Computational Structure of the self-organizing Controller

The overall behaviour of the robot is influenced by the following *instincts*, the first two achieving the 'obstacle avoidance' competence, the third instinct in conjunction with the first two achieving 'wall following' behaviour:

1. Do not have your whiskers bent
2. Move forward, whenever possible,

   and, only for 'wall following' behaviour,
3. Do touch something after a while.

These instincts are part of the *monitor* which decides whether an action was successful (instincts satisfied) or not (instincts not satisfied).

For **obstacle avoidance** competence the robot behaves as follows: as long as no sensory signals are received ('no obstacle ahead') the second instinct will make the robot move forward. As soon as either of the whiskers detects an obstacle, however, a conflict arises with the first instinct. The robot will therefore try the action that is stored in its associative memory for that particular constellation of input signals. This action will be performed for a while, typically about five seconds. If the action was successful (i.e. the instincts are satisfied) all is well and a forward move is again possible. If, however, the input signals still indicate an obstacle (and this might well be an obstacle other than the original one - the robot might have moved into more trouble) something else has to be tried. At this point a generate-and-test procedure is used, similar to the 'associative search' of Barto and Sutton [[Barto *et al.* 83]]. The move associated with the output node that is next strongest in excitation is selected and performed for a period of time. After each unsuccessful attempt to satisfy all instincts this period of time is extended because we assume that unless the robot moves clear of the obstacle it might have moved into even deeper problems. If, by following this method, a successful move has been found, this move is stored in the associative memory. The next time the same set of input signals is detected this newly learned move will be performed, and hopefully be successfull immediately. Teaching the associative memory is another task of the monitor. (The version of the net we have used has three input units (for three sensors) and four output units. The gain η is 0.2 .)

Obviously with two motors that can move either forward, backward or be switched off there are nine possible robot actions. The chances of selecting a 'good' move out of this set are smaller than picking a 'good' move from a restricted set. Such a restricted set could comprise the following actions: forward, backward, swift left and swift right. We have used this restricted set of robot actions.

For **wall following behaviour** the robot, using all three instincts mentioned above, behaves like this: Starting, the robot obeys the first instinct and moves forward. However, after a fixed period of time (about 4 seconds) the third instinct is no longer met, the robot therefore tries to touch something (the wall). In order to be able to do this the robot has to learn on which side the wall is to be found. As soon as the wall is touched, it is the second instinct that is no longer satisfied, and the robot has to get away from the wall. So, for wall following behaviour the robot has to learn two things: First of all, on which side the wall is, and secondly, how to get away from it.

B.4.3 Experimental Results

When switched on Alder starts moving forward, as long as there is no obstacle in the way. It is thus obeying the second built-in instinct. As soon as an obstacle is detected, experience comes in: Alder tries the learnt move to get away from the obstacle. If this action proves to be successful, the robot will store it in its internal memory, otherwise yet another action will be tried. We observed that if the robot is left for some time, trying out moves and learning the good ones, the robot eventually behaves very efficiently.

Without hesitation it turns away from an obstacle, usually using the action that will achieve this the quickest way.

In the case of the robot learning to follow a wall, something similar happens: Initially, Alder's behaviour is not at all goal oriented, actions are selected without reasoning based on previously obtained knowledge. Very quickly[3], however, Alder learns which actions will satisfy its three built-in instincts and the robot will follow the wall successfully, moving alongside the wall in a zig-zag manner.

There are a few very interesting side observations. For example: it doesn't matter at all which way the sensors are connected[4], or the motors (as long as sensors can be read and motors switched on or off, of course). Because the system is self-organizing it learns what to do, regardless of whether a motor is plugged in 'a particular way round' or not. The same applies to the sensors: whether a sensor is read on the most significant bit, or the least, or whether it is an infrared sensor or a whisker, does not matter. Furthermore it does not even matter how many sensors are actually plugged in. The controller will make use of all the information available. We also observed that if a learnt reaction was no longer successful (for example because one of the motors had failed or because the polarity of the wires had been reversed), the control algorithm would find new solutions to comply with the built-in instincts[5].

Due to the plasticity built in the controller the robot is able to adapt to a changing environment, too. For example, the robot will discover the best behaviour for escaping a dead end (which is to turn in one and the same direction every time either of the front whiskers is touched), even if it entered the dead end with the 'wrong' sort of behaviour (for example the obstacle avoidance behaviour, i.e. turning away from a touched whisker). Similarly, the robot will successfully re-learn the wall following behaviour if it is turned round (which, from the robot's point of view, means that the wall is now "on the other side").

B.5 Conclusion

We have designed a self-organizing controller for a mobile robot. *Competence generators*, governed by instincts, generate the necessary competences (a synthetic rather than an analytic approach). We have built a mobile robot which serves as a first experimental platform. The controller of this robot is equipped with a linear net as a memory unit and a monitor which provides teaching signals for the memory.

[3]Not more than three trials - this can be influenced by setting the gain appropriately.

[4]i.e. where left, right and further sensors are plugged into the interface.

[5]Always, of course, within the set of possible actions!

Acknowledgements

The work reported here is supported by a grant from the UK Science and Engineering Research Council (grant number GR/F/5852.3). We would also like to thank Peter Forster, our colleague on RUR, for helpful comments on earlier drafts of this paper.

Appendix C

Mapbuilding using Self-Organising Networks in "Really Useful Robots"[1]

Ulrich Nehmzow and Tim Smithers[2]

Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill
Edinburgh EH1 2QL
Scotland
Telephone 031 - 667 1011 ext 2529/2517
Telex 727442 UNIVED G
E-mail:ulrich@uk.ac.ed.edai tim@uk.ac.ed.edai

September 1990

[1]published in: Jean Arcady Meyer and Stewart Wilson (eds.), *From Animals to Animats*, pp. 152–159, MIT Press Cambridge Mass. and London, England, 1991.

[2]Names appear in alphabetical order, with both being principal authors on this occasion.

C.1 Abstract

The Really Useful Robots (RUR) project is seeking to understand how robots can be built that develop and maintain the task achieving competences they require for flexible and robust behaviour in variable and unforeseen situations, as opposed to these being installed by their designers. In this paper we present an experimental autonomous robot with a map building competence which uses a self-organising network. Map building forms a necessary step on the way to development of a navigational competence. Some encouraging initial test results are also presented.

C.2 Introduction

The traditional approach to control in (mobile) robots is to decompose the task into separate components, and implement these using standard control techniques, see [Levi 87], for example. This we call an *analytical approach*. Alternatively, a control structure can be built 'bottom up', first building foundational competences (such as 'move around and avoid obstacles'), and later on top of these more complicated competences (such as 'explore', 'map building', and 'map using'). This we call a *synthetic approach*, see [Brooks 85], for example.

At Edinburgh we have adopted a synthetic approach in what we call the 'Really Useful Robots' project (RUR) [Nehmzow *et al.* 89]. This project is attempting to develop a control architecture which supports the development of task achieving competences by the robot. In other words, we are trying to understand how a robot can sequentially acquire and maintain the behavioural competences it requires, rather than have them 'installed' by us as its designers. We believe that this *autonomous acquisition* of task achieving competences will lead to greater flexibility and robustness in the behaviour of robots with respect to variable and unforseen situations. In investigating this idea we are motivated and informed by the adaptive control mechanisms we see in simple animals which result in them having flexible, reliable, and robust competences well matched to the tasks they are responsible for achieving and to the environment in which they are exercised.

Trying to get a robot to acquire the skills it needs means that as many decisions as possible are left to the robot, rather than being predefined by the designer. *Alder*, the first of the 'Really Useful Robots' (see figure B.6)[3],

[3]Alder is a mobile robot whose base is built with a Fischertechnik kit. It is about 25cm long, has an 8052 based microcomputer on board (16k RAM) and is

is able to adapt to a changing environment, and to acquire useful competences. It uses what we call *fixed* and *plastic* components in its control architecture to achieve this, fixed components being the mechanics of the robot, its shape (morphology), the control program, and its so-called 'instincts', the plastic component being an artificial neural net (in the current system). So far we have successfully demonstrated robot acquisition of simple obstacle-avoidance, dead-end-escape, and wall-following competences. The next stage is to get the robot to develop a navigational competence, using a similar approach. This first requires a mapbuilding competence.

As in *controlling* mobile robots, so in *mapbuilding* a traditional approach can be taken. The traditional approach either uses some kind of fixed structure that is filled with information as it is obtained by sensory activities,see, for example, [De Almeida & Melin 89], [Moravec 88], [Goto & Stentz 87] or [Tsai *et al* 90]. We call these 'self detailing schemes'. Alternatively, a complete 'ready-made' map may be provided from the start, see, for example, [Tsubouchi and Yuta 87], which we call 'pre-installed map schemes'. In these traditional approaches the structure of the map is defined by the *designer*, and so is not necessarily the best to store and retrieve the knowledge the *robot* needs to have and can acquire about the world. Here, as well as in controlling robots, *predefining means limiting.* Limiting what is represented by a map is not of itself a problem, but knowing what needs to be represented and what doesn't is. To get around this problem we think that the process of constructing a map should be left, as far as possible, to the robot, not to the designer, the idea being, that a map built by the robot is likely to be more useful to it, than one given to it by its designer.

Alder uses a self-organising network (see [Kohonen 88]) to construct internal representations of the world it experiences as it moves around. In this paper we describe how this mapbuilding competence is implemented and some early test results which we believe are encouraging.

C.2.1 The place of this paper within the Navigational System of Alder

The plan is to design a navigational system as shown in figure B.1.

Fixed components (as mentioned above) within this structure are the sensors, including the higher level interpretation of incoming data (the top block in figure B.1), and the behaviour[4] of the robot. *Plastic components* are the self-organising network, as described in section B.3, and an artificial neural network which associates particular robot movements in the real

equipped with up to seven tactile sensors plus odometer. In addition to this a sonar sensor is available, but has not been used to obtain the results presented in this paper. More information about Alder and the 'Really Useful Robots' approach can be found in [Nehmzow *et al.* 89].

[4]'Behaviour' here means any fixed pattern of motor-sensory activity, it is therefore a 'fixed' component of the robot.

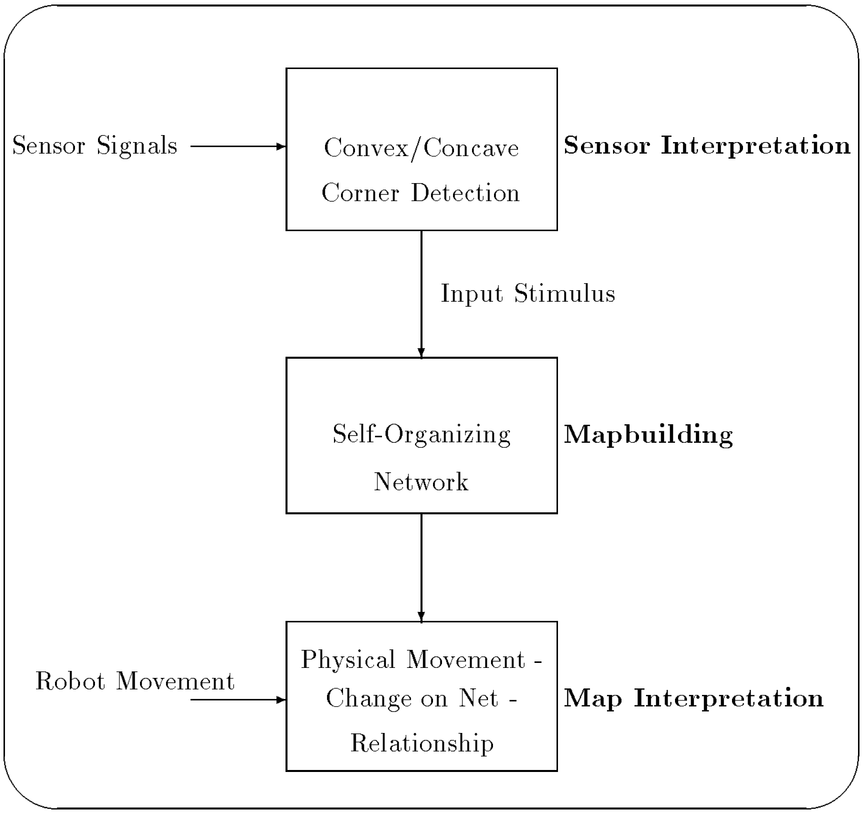

Figure C.1: The proposed Navigational System of Alder

world with changes in excitation of the self-organising network (the bottom block in figure B.1). The combination of all these components forms the navigation system.

This paper discusses only one aspect of this system: mapbuilding, using self-organising networks (the middle block of figure B.1).

C.2.2 What is a map?

We take a map to be *any one-to-one mapping (bijection) of a state-space on to map-space*. Examples of such maps are the plan of the London Underground, the Edinburgh telephone directory, and a family tree. The connections between stations, phone owners and numbers, and family members are represented in all these cases. A map is therefore not just a representation of a 'birds-eye-view' of the world the robot inhabits.

In our RUR project the state-space that is represented using a self-organising network has nothing to do with actual physical locations in the real world, at least not directly. What *is* represented by these maps is the result of previous and present motor-sensory experience of the robot.

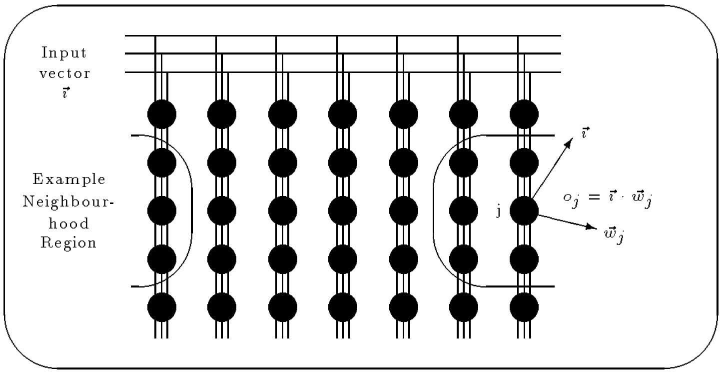

Figure C.2: A two-dimensional self-organising network

The motor actions and sensory actions of the robot are tightly coupled, so tightly coupled in fact that they cannot be viewed independently of each other! Therefore the state space represented *does* in the end have something to do with actual physical locations of the robot in the real world, but only indirectly through motor-sensory behaviour of the robot.

Whenever the term 'map' is used in this paper it is used in the sense of the definition given above, i.e. a bijection of state-space on to map-space.

C.3 Self-Organising Networks

Unsupervised learning can be achieved, using self-organising networks [Kohonen 88]. These networks develop internal representations of the input space by mapping distinct input vectors onto distinct areas of the network (injection). One advantage of these networks is that they can accept redundancy in the input vectors. Redundant information is ignored by the net as long as the input vector contains *sufficient* information for a meaningful mapping to be generated. Surplus information does not impair performance. This fact makes it easier to define a suitable input space. Figure B.2 shows an example of a two-dimensional self-organising network.

The input vector $\vec{\imath}$ is the same for all the cells. The output o_j of cell j is determined by the scalar product of input vector $\vec{\imath}$ and the weight vector $\vec{w_j}$ of cell j, and is given by:

$$o_j = \vec{w_j}\vec{\imath} \tag{C.1}$$

where $\vec{w_j}$ is the individual weight vector of cell j. The weight vectors are unit vectors: $||\vec{w_j}|| = 1$.

Initially, the elements of the weight vectors are set to randomly selected values such that they are all unique. Therefore one cell of the network is bound to respond most strongly to a particular input presented to the net. The weight vector of this maximally responding cell, as well as the weight vectors of all the cells within a defined neighbourhood around this cell, are changed according to the following equation:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \eta(\vec{\imath} - \vec{w}_j(t)) \tag{C.2}$$

where η is the so-called 'gain', a value that determines the amount of change (typically around 0.5). Weight vectors outside the specified neighbourhood [5] remain unchanged.

After several 'epochs', i.e. presentations of input vectors to the network, typical dissimilar responses appear for dissimilar input vectors. Thus a mapping arises, whereby different input vectors are mapped onto different regions of the network (bijection). This therefore is a *map* according to the definition given in section B.2.2.

C.3.1 The Self-Organising Network used for Mapbuilding in Alder

Obviously, border effects can occur if the network has edges. This can be avoided by joining the opposite edges of the network. In the case of a two-dimensional network this yields a torus-shaped network.

As long as a network with a sufficient number of cells is used [6] it is not necessary to use a network of the same dimensionality as the space the agent acts in (i.e. although the robot moves in a two-dimensional world, a one-dimensional network can be used to obtain the self-organising state-space-representation). Since the computation for a one-dimensional network is much less [7], and the computational power of Alder is limited, we used a ring of 50 cells as self-organising network (see figure B.3).

The behaviour of this network is as previously described in section B.3, the neighbourhood within which weight vectors are updated is $\pm$ 2 cells (constant over time). A typical response of the network is shown in figure B.4 (in this figure the ring is cut and shown as a line).

The particular response of the network to a particular input stimulus (i.e. the activity pattern of all the cells of the net) can itself be viewed as a vector. This is convenient for later analysis, the particular response vector to a particular vector is called $\vec{\kappa}$ in this paper.

[5] The neighbourhood size is often chosen to be decreasing over time, in order to achieve quick changes in the beginning, and only little changes later.

[6] A rough calculation to determine the minimum number of cells required: If the network is to represent n distinct states, and the neighbourhood size within which weight vectors of cells are updated is m, then the minimum number of cells obviously is n $\cdot$ m. However, this is the *minimum* number, in practice a bigger number is needed.

[7] The number of necessary weight-vector changes is proportional to n^m, where n is the neighbourhood size and m the dimensionality of the map.

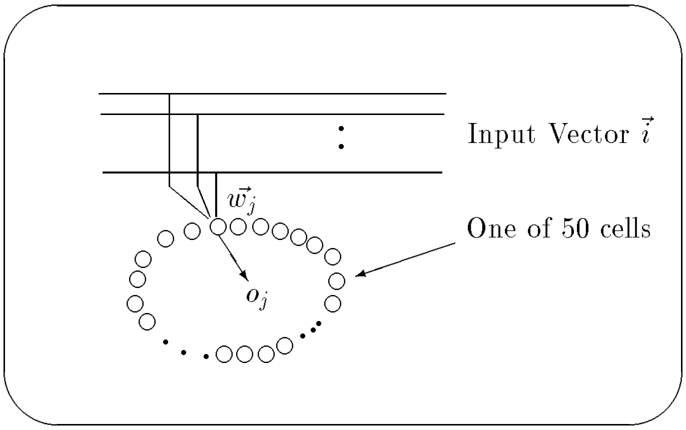

Figure C.3: The self-organising network used on Alder

C.3.2 The Role of Robot Behaviour

Environment, agent (that is robot), and task are very tightly coupled and cannot be treated individually. This became obvious through the following observation:

The first behavioural pattern the robot showed was to wander around randomly and simply avoid obstacles. Distances travelled between obstacles and sensor readings obtained were used as input vectors to the net (see also section B.3.3). The result was that the $\vec{\kappa}s$ of the net in the settled state showed no strong correlation to particular physical locations. This is not surprising since if *we* were to wander randomly around a darkened room avoiding obstacles encountered by feeling them, we too would not build up a good 'picture' of the room and what is in it and where.

When we changed the robots behaviour to a wall-following behaviour the results became much more interesting: together with an improved input vector (see B.3.3) the $\vec{\kappa}s$ did indeed show a correlation with particular physical locations in the real world. By adopting wall-following behaviour the robot negotiated its way around the perimeter of its enclosure and the $\vec{\kappa}$s corresponded to corners.

C.3.3 The Input Vector

A self-organising network clusters the input information presented to it in a (statistically) meaningful way. If the input to the net contains no meaningful information (meaningful in relation to the task of constructing a state-space representation of the robot in its world, that is) the network will not develop any meaningful structure. The very first input vector we used simply contained information about whether the robot had "seen"

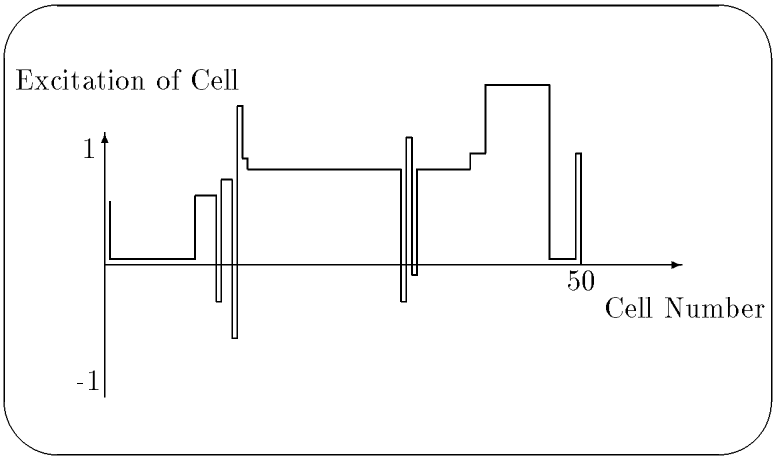

Figure C.4: A typical response of the ring after having settled in

something on its left or on its right hand side, plus information about two previous sensor readings (again only whether the obstacle was seen on the left or on the right), plus odometry information. Such information is too weak to construct a meaningful state-space representation (at least without requiring a great many input vectors), consequently the $\vec{\kappa}s$ in the settled state of the net had little correlation with particular locations in the real world.

This was obviously due to a lack of sufficient structure in the input vector presented to the net. We therefore enriched the input vector by preprocessing the sensory information obtained. Instead of taking data straight from the sensors and feeding it into the net, sensor readings were used to detect convex and concave corners[8]. This information (whether a corner was encountered, and whether this corner was convex or concave) was then used as input to the net. The input vector eventually used to obtain the results shown in section B.4 contained information about the present corner and previous corners encountered as well as the distance travelled between the present and the previous corner (An example of such an input vector is shown in figure B.5)[9].

[8]This is very easily achieved: If the time the robot needs to turn towards the wall exceeds a certain threshold time, then it is assumed that a convex corner is detected. Similarly, if the time it takes the robot to get away from a detected obstacle exceeds a certain threshold time, it is assumed that a concave corner is detected.

[9]Note that it is necessary to encode the information about convex and concave corners using 2 bits, because artificial neural networks can only make use of non-zero input lines (see also equation B.2).

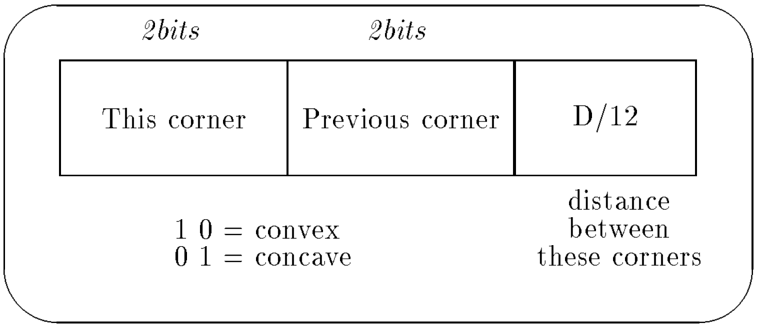

Figure C.5: A typical Input Vector

C.3.4 A brief summary of the whole mechanism

1. Initialise the self-organising network (i.e. the ring) by filling the weight vectors of all cells with randomly chosen values.

2. Normalise all the weight vectors.

3. Present an input stimulus to the ring (the input stimuli chosen for our experiments are desribed in section B.3.3).

4. Determine the response to this stimulus for each cell of the ring according to equation (B.1).

5. Determine the unit that is responding most strongly.

6. Update the weight vectors of the five units within a neighbourhood of $\pm$ 2 cells of the maximally responding cell, according to equation (B.2).

7. Normalise those five weight vectors again.

8. Continue with step 3.

C.4 Experimental Results

The experiment was conducted as follows: The robot was placed within a fence (see figure B.6) and allowed to explore it, following the wall for several rounds.

Whenever a convex or concave corner was detected an input vector as described in section B.3.3 was created and presented to the ring. The gain η for updating weight vectors (see equation B.2) was initially very high (5.0) and decreased by 5 per cent after every presentation of an input vector. The longer the robot went around in the enclosure, the more settled the

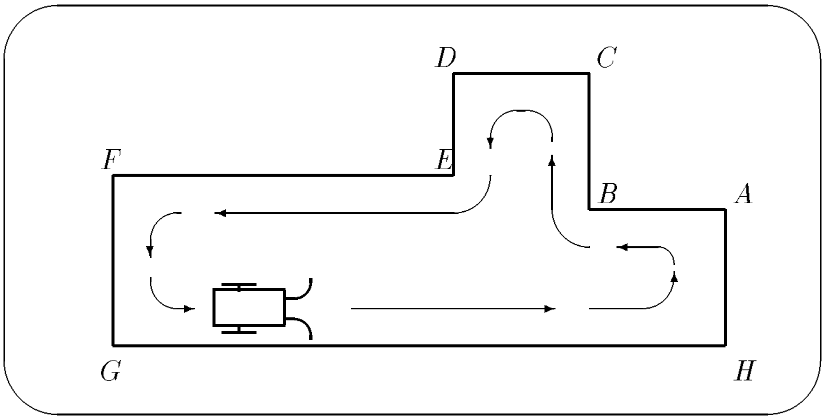

Figure C.6: A typical environment for Alder

ring became and the more precise the response to a particular input stimulus. After about three rounds a particular corner was marked by pressing the 'attention' button on the robot. The $\vec{\kappa}$ of the ring for that particular ('marked') corner was then stored ($\vec{\kappa_m}$), all subsequent $\vec{\kappa}s$ were compared with $\vec{\kappa_m}$ by calculating $||\vec{\kappa} - \vec{\kappa_m}||$; the smaller this value, the closer the $\vec{\kappa}$ at that particular corner to $\vec{\kappa_m}$. Obviously, if the robot is able to construct a meaningful internal state space representation this difference should be small when the robot is at the marked corner, and it should be noticeably larger at any other corner.

The input vector chosen will determine the quality of the internal representation obtained. Only information that is actually contained in the input vector sequence can eventually be found on the internal map! The results we obtained highlight this very clearly:

Originally we used an input vector that contained information about the current corner encountered and the previous corner to that one, as well as the distance travelled between these two corners (this vector is shown in figure B.5). In figure B.7 we can see that corner H is uniquely identified, using this input[10].

It is obvious that corner H is the easiest of all corners to recognize, because of the long distance travelled between corners G and H.

Looking at corners C and F, on the other hand, one might expect that these corners get confused on the internal map, because both C and F are concave corners, the previous corner in both cases is convex, and the distances between them is similar. Exactly this is observed (see figure B.8).

The same applies to corners B and E: they are both convex, both have

[10]The bars in these diagrams show $||\vec{\kappa_m} - \vec{\kappa}||$. The smaller this value, the closer the response of the net to the response the net showed at the marked corner. The bigger this difference, the more distinct are the marked corner and the corner currently encountered. In other words: the smaller the value, the more alike are

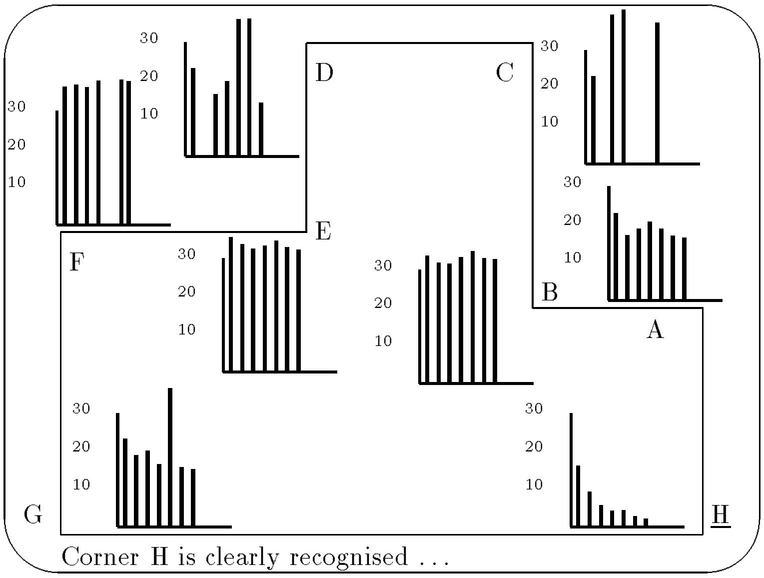

Figure C.7: Recognizing corner H, looking at one previous corner

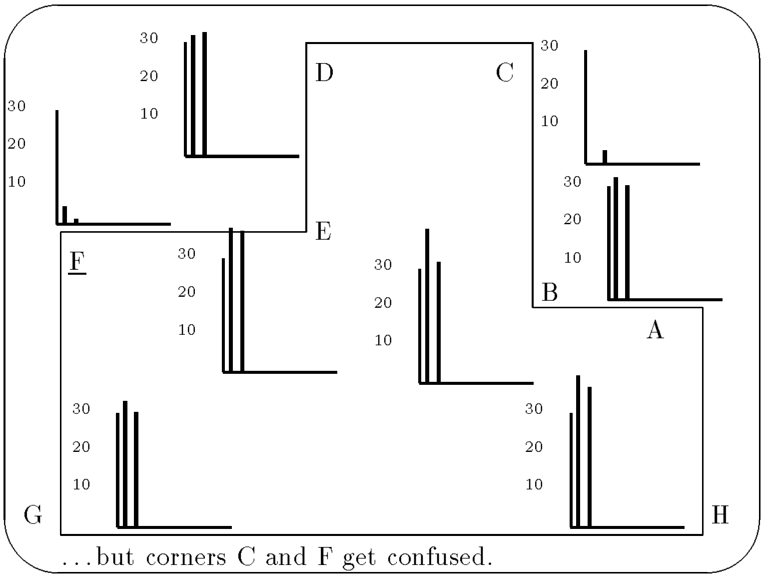

Figure C.8: Recognizing corner F, looking at one previous corner

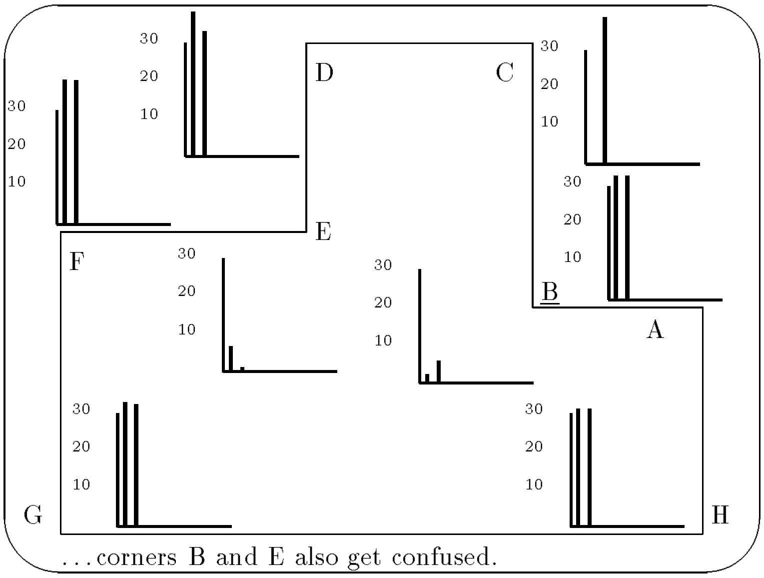

Figure C.9: Recognizing corner B, looking at one previous corner

previous corners that are concave, and again the distances between them are similar. Again, our expectations that these two corners might have an identical representation on the resulting map are experimentally verified (see figure B.9). This confusion of corners C and F and B and E respectively arises from the fact that the motor-sensory-space of the robot (input-space of the network) is not well coupled to the physical space the robot is acting in. According to the definition given in section B.2.2 the resulting network is indeed a map, but a map of the motor-sensory space rather than the physical space[11].

The obvious answer to this problem is to increase the information contained in the input vector. Taking the example of corners B and E, even looking at *two* previous corners is not going to avoid confusion of these two corners, because for both corner B and E the respective previous two corners are of the same type, and the distances travelled are similar too.

To verify our theory we nevertheless used an input vector that contained information about the present corner as well as *two* previous corners: as expected corners B and E still cannot be distinguished (figure B.10), and corners H gives no problems, as before (figure B.11).

Only if the input vector contains information about the present corner as well as the *three* previous corners do corners B and E have different input vectors, therefore one would expect the map to be able to distinguish them.

marked corner and present corner.

[11]The mapping of the physical space onto map-space here is an injection, that of the motor-sensory space a bijection.

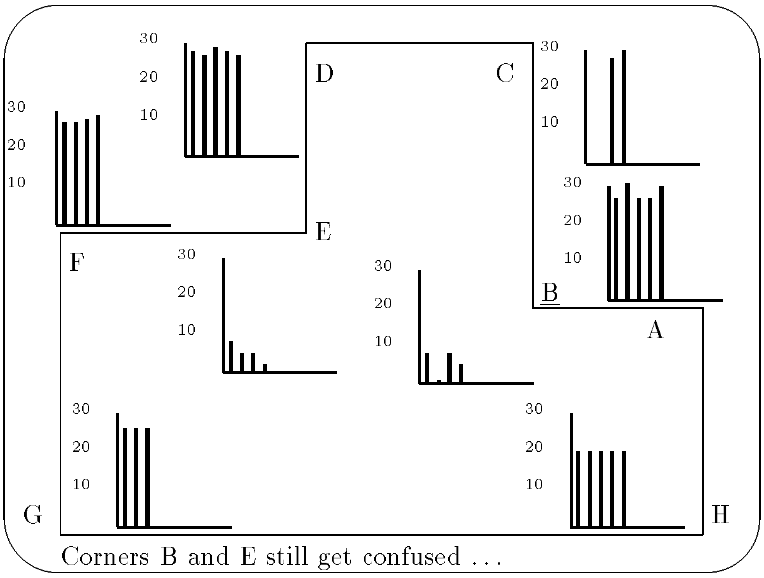

Figure C.10: Recognizing corner B, looking at two previous corners

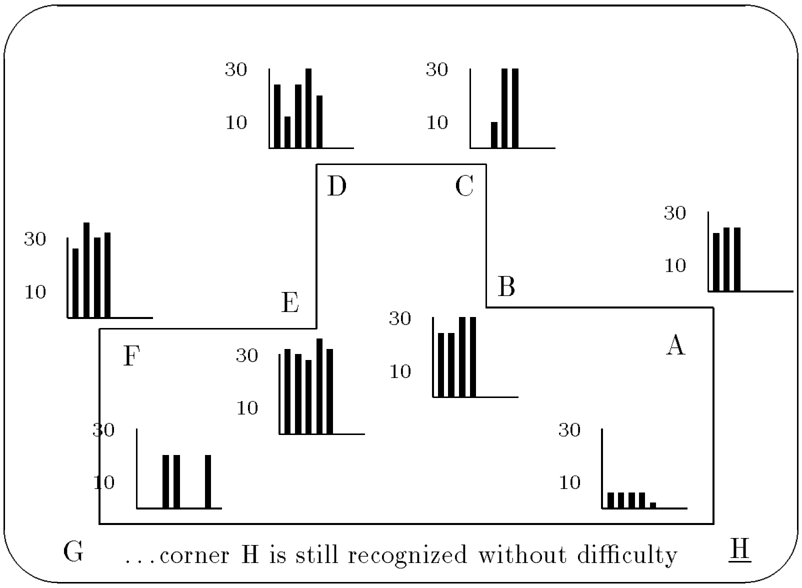

Figure C.11: Recognizing corner H, looking at two previous corners

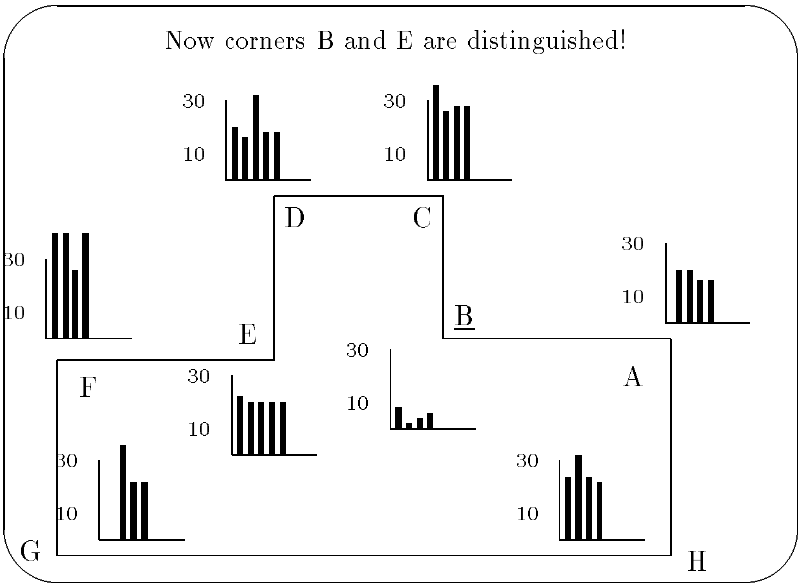

Figure C.12: Recognizing corner B, looking at three previous corners

This is exactly what happens (see figure B.12)!

Again, as in all the previous cases, corner H is recognized without any problems (see figure B.13).

These observations show the relationship between input vector and resulting internal representation very clearly, they also show that self-organising networks *can* be used to generate meaningful internal representations of the world.

Another observation we made is that the 'certainty' increases with increasing experience: $||\vec{\kappa_m}(t) - \vec{\kappa_m}(t+1)||$ decreases continuously as the robot's experience increases (see figure B.14).

In summary, this is what happened: Having had sufficient time to explore the environment, the robot was able to recognize particular corners that had been marked by the experimenter, provided this was theoretically possible[12]. In cases where the input vector presented to the self-organising net contained insufficient information, Alder confused corners that looked alike, simply because their input vectors were (almost) identical. This is not surprising. The robot's ability to recognize the marked corner reliably increased with experience, $||\vec{\kappa_m}(t) - \vec{\kappa_m}(t+1)||$, the difference between excitation of the net at the marked corner at time t and excitation at the marked corner at time t+1, decreased continously. It eventually became zero.

[12]Alder signals the recognition of a marked corner by a swift turn, first to the left and then to the right.

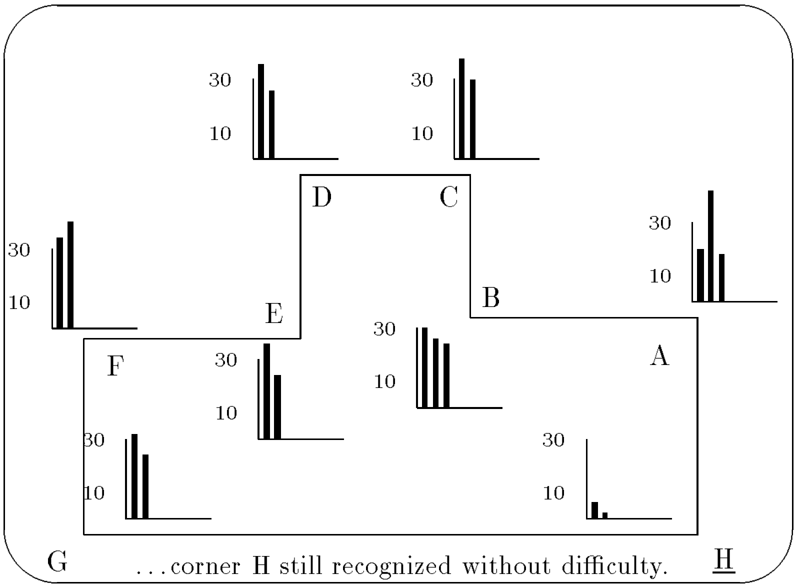

Figure C.13: Recognizing corner H, looking at three previous corners

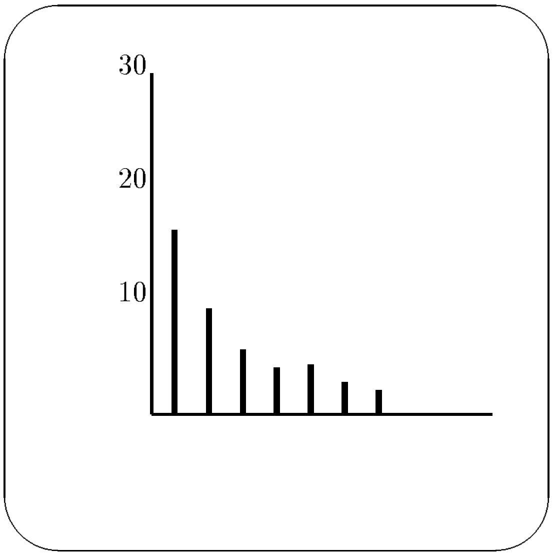

Figure C.14: $||\vec{\kappa_m}(t) - \vec{\kappa_m}(t+1)||$

C.5 Comparison with Bee Navigation

As [Cartwright & Collett 83] have found in their experiments, *Apis mellifera* use nearby landmarks to guide their way to a food source. "[Bees] do not find their way using anything analogous to a floor plan or map of the spatial layout of landmarks and food source. The knowledge at their disposal is much more limited, consisting of no more than a remembered image of what was present on their retina when they were at their destination. ... Bees find their way, the experiments seem to say, by continuosly comparing their retinal image with their snapshot and adjusting their flight path so as to lessen the discrepancy between the two." We do not claim that the mechanism used for landmark learning in Alder is identical to the one used in bees; however, there are similarities. Like bees, Alder does not generate a floor plan (a conventional map), instead it uses 'snapshots' (distinctive excitation patterns of the self-organising network as a response to the sensory input) to recognize locations.

Another similarity to bee navigation is the fact that the mechanism described in this paper is very robust and fairly immune to noise. 'Undetected' corners, varying distance measurements and even moving the robot to a different location (without the control algoithm realising this) do not impair Alder's ability to recognize locations. [Cartwright & Collett 83]: "The bee's guidance system is immune to a considerable amount of noise".

C.6 Conclusion

Building an internal state space representation (a *map*) is an essential step towards developing a navigational competence in a mobile robot. We have shown that it can be achieved, using self-organising networks. Particular input stimuli result in distinct reponses of the (ring-shaped) self-organising network, distinct physical locations in the real world can thus be distinguished, provided that robot behaviour and sensing are well coupled to the environment and the task of mapbuilding. The mechanism presented is similar to the 'feature recognition' mechanism used by bees to navigate—bees compare stored retinal patterns to the current retinal pattern to calculate a flight direction vector ([Gould & Gould 88] [Waterman 1989] and [Cartwright & Collett 83]).

The next step in the RUR project is to develop a network that can learn the relationships between physical movements in the world and observed activity changes in the self-organising network (the bottom block shown in figure B.1), which will make true navigation possible.

Acknowledgements

The work reported here is supported by a grant from the UK Science and Engineering Research Council (grant number GR/F/5852.3). We would also like to thank John Hallam and Peter Forster, our colleagues on RUR, and Mitch Harris for helpful comments on earlier drafts of this paper.

Appendix D

Location Recognition in a Mobile Robot[1]

Ulrich Nehmzow, Tim Smithers and John Hallam

Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill
Edinburgh EH1 2QL
SCOTLAND

E-mail: ulrich@uk.ac.ed.edai

[1]Published in: Günther Schmidt (ed.), *Information Processing in Autonomous Mobile Robots*, Springer Verlag, Berlin, Heidelberg, New York, 1991

D.1 Abstract

Self-organising structures are a dominant feature of the experimental mobile robots built in our "Really Useful Robots" project. This paper continues where [Nehmzow & Smithers 91a] finished. It explains some initial experiments using self-organising feature maps, and how those maps can be used by a mobile robot to recognise locations in its environment. This location recognition capability is achieved without using sensory information. Instead information derived from the motor actions of the robot is used, and shown to be sufficient.

D.2 Introduction

Self-organising feature maps (SOFMs), [Kohonen 88], can be used for simple navigation tasks. In earlier work, [SAB 91], we have shown that "Alder", the first of the "Really Useful Robots" (RUR) (see page 74), is able to use a SOFM to recognize particular locations in a simple enclosure after it has had sufficient time to explore and to "learn" about this enclosure. In these earlier experiments the input that was presented to the SOFM was derived from sensor signals and contained explicit information about "landmarks" that the robot encountered, and the distance travelled between them. These landmarks were the concave and convex corners in the robot's enclosure.

In the RUR project, we are investigating ways to make mobile robots more flexible, and in particular, to make them more able to cope with *unforeseen situations*. We believe that in order to achieve this as many decisions as possible should be left to the robot; in particular decisions about what and how information derived from sensors or other sources internal to the robot should be interpreted for the purposes of control in task achieving behaviour. For the same reason we aim to equip the robot with as little predefined knowledge about its environment as possible [Nehmzow *et al.* 89]. Information concerning landmarks is specialised information. Therefore, we decided to try to achieve similar results to those reported earlier, but this time using less specific input information. Whereas before we used processed sensor information (denoting corner types) to achieve location recognition, we now use *no sensor information* at all! Instead we use information about the history of the *motor action commands* of the robot controller. In this paper we describe how we achieved this, again using self-organising networks.

D.3 The Duality of Sensing and Acting

Sensing and acting are typically treated as separate functions in robotics. We believe, however, that sensing and acting are two aspects of the same function, and that they therefore cannot be successfully analysed in isolation. The actions of a robot, just like those of a person, determine to a large extent the sensory signals it will receive, which will in turn influence its actions. Breaking this tight interaction into two separate functions leads, we believe, to an incorrect decomposition of the robot control problem. While particular features of a robot's sensors and actuators do play an important role in determining its performance, these effects cannot be determined by their separate analysis. Acting and sensing have to be seen together; neither acting nor sensing alone will make the agent succeed.

The input vector we have chosen for the following experiments demonstrates this point: it contains no direct information about sensory input. The information it does contain is derived from the motor action commands of the robot controller, but these, as we have said, are themselves influenced by the sensory signals received by the robot as a result of its actions. We chose to use information derived from the motor action commands of the robot controller, rather than from sensor signals, because they form a smaller set of signal types, they are much less subject to noise, but they still adequately characterise the interactions between the robot and its environment as it seeks to achieve its task—wall follow, in our case.

For the location recognition experiments the robot is placed in an enclosure as shown in figure C.1; it then follows the wall using a preprogrammed wall-following and obstacle-avoidance behaviour (alternatively these skills could be acquired by learning, see [Nehmzow *et al.* 89]). In other words, the robot is governed by its preprogrammed wall following behaviour which, of course, does use sensory information. The process of constructing the SOFM is, however, independent of the wall following behaviour, it simply "looks" at the motor action commands issued by the controller as the robot performs this wall following task.

Every time a new motor action command is issued, that is, every time the wall-following or the obstacle-avoidance behaviour forces the robot to change direction a motor action vector is generated. This is a nine bit vector which contains information about the state the motors were in until this change, and thus the direction the robot has been travelling up to this moment (forward, left or right)[2], as well as information about how long it was in this state (see figure C.2). This motor action vector forms the input to the SOFM.

Thus, from figure C.2 we can see that no information concerning sensor

[2]Strictly speaking, they contain information about the last command from the robot controller. Whether this command was actually obeyed by the robot or not is not sensed.

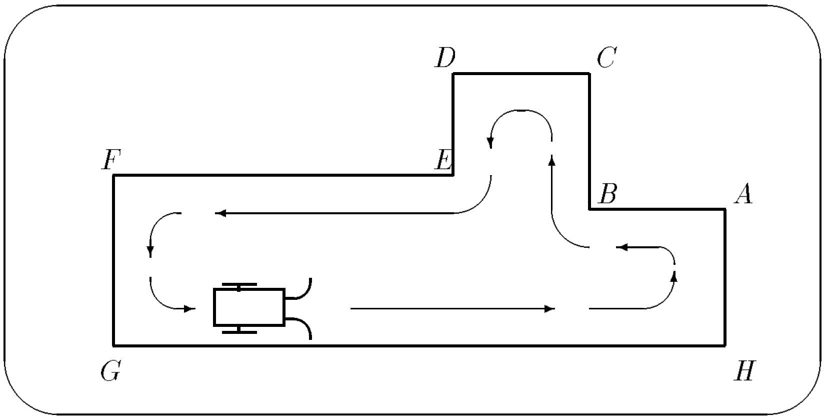

Figure D.1: A typical environment for Alder

Motor Action		**Duration**
Forward	01 01	00000 less than 0.9s
Left	01 10	00001 0.9 - 1.3s
Right	10 01	00011 1.3 - 1.7s
		00111 1.7 - 2.1s
		01111 2.1 - 2.6s
		11111 over 2.6s

Figure D.2: The motor action vector

signals is directly presented to the SOFM. The only information available to the network concerns motor action commands.

D.4 Early Experiments

D.4.1 Self-Organising Feature Maps

The mathematics for the kind of self-organising feature maps (SOFMs) that we use can be found in [Kohonen 88] or in [Nehmzow & Smithers 91a]. Basically, they conform themselves to the structure of the space of input signals, clustering input signals in such a way that related inputs excite neighbouring areas of the network (they are *topology preserving*), and so that the relative sizes of these areas reflect the relative probabilities of the different inputs in the data presented. By making use of these two properties of SOFMs they can be used to perform a kind of unsupervised learning.

To investigate how this might be done using only data derived from motor commands we first carried out the following experiment in which we used a two-dimensional SOFM of ten by ten cells.

D.4.2 Experiments with a 10 x 10 SOFM

The robot was left to explore its enclosure (figure C.1) by following the walls, generating input vectors as shown in figure C.2 each time a new motor action command was issued by the robot controller for any reason. The following pictures show the response of a ten by ten cell SOFM to different input stimuli; the terms “right”, “left” and “forward” describe the motor action command that was performed, followed by a number that indicates the length of time for which this action was performed (see figure C.2).

```
------------
|.       ..|
|..   .....|
|//.  .....|
|//.  ...  |
|///.....  |
|++///..   |
|*+//.     |
|++/..     |
|..        |
|..        |
------------
Forward, zero
```

```
------------
|  .... ...|
|  /..///..|
|++/..///..|
|*+/..///  |
|+++/////  |
|///////.  |
|/////...  |
|/////.....|
|  ........|
|  ........|
------------
Forward, one
```

```
------------
|  .....///|
|  ..//////|
|//////////|
|////////..|
|//+++///..|
|//+*+///..|
|//+++//...|
|./+++/////|
|  ..././//|
|  ....////|
------------
Forward, two
```

```
------------
|   ../////|
|  ../+++//|
|/////+*+//|
|/////+++..|
|///+++++..|
|.//++++/..|
|.//+++////|
|..////////|
|  ...///..|
|  ..../...|
------------
Forward, three
```

```
------------
|   ..//+++|
|  ...+++++|
|//../+++++|
|/////+++//|
|/////+*+//|
|..///+++//|
|..////////|
|..////////|
|     //...|
|     .....|
------------
Forward, four
```

```
------------
|    .//+++|
|  ...//+++|
|....////+*|
|....//////|
|..///+++//|
|..///+++//|
|..////////|
|..///./...|
|     .....|
|     .....|
------------
Forward, five
```

```
------------
|          |
|          |
|          |
|        ..|
|        ..|
|        ..|
|.    .....|
|..........|
|++////..//|
|*+////.///|
------------
 Left, zero
```

```
------------
|          |
|          |
|..    ..  |
|..    ....|
|.....  ...|
|  ........|
|  ...////.|
|..////////|
|//+*+/////|
|//++++////|
------------
 Left, one
```

```
------------
|       ...|
|     .....|
|..  ......|
|... ....//|
|..///...//|
|  ///..///|
|  ////////|
| .///+++++|
|/////+++*+|
|/////+++++|
------------
   Left, two
```

```
------------
|     .....|
|     ///..|
|.. ..///..|
|...../////|
|....//////|
|  ..//////|
|  ../+++++|
|  ///+*+++|
|..///+++//|
|..////++//|
------------
  Left, three
```

```
------------
|     ..///|
|     /////|
|    ./////|
|    .///++|
|  ...///++|
|  ...//+++|
|  .../++*+|
|  /.//++++|
|..////+///|
|..////////|
------------
  Left, four
```

```
------------
|     ..///|
|     .////|
|    ...///|
|    .///++|
|  ...///+*|
|  ...///++|
|  ...//+++|
|  .../////|
|...../////|
|...../////|
------------
  Left, five
```

```
------------
|*++//..   |
|++///     |
|  //.     |
|  ...     |
|          |
|          |
|          |
|          |
|          |
|          |
------------
 Right, zero
```

```
------------
|//+*+//.  |
|//+++...  |
|..+//...  |
|..///...  |
|.....     |
|   ...    |
|   ...    |
|   ...    |
|          |
|          |
------------
Right, one
```

```
------------
|///++//...|
|//+++//...|
|..+*+/....|
|..+++/..  |
|..///...  |
|  ///...  |
|  ///..   |
|  ...     |
|          |
|          |
------------
 Right, two
```

```
------------
|..///++/..|
|../+++//..|
|../+*///..|
|../++///..|
|.../////..|
|  ...//...|
|  ........|
|  ........|
|     ...  |
|      .   |
------------
Right, three
```

```
------------
|..///+*///|
|..////////|
|  //+/////|
|  //////..|
|  ...///..|
|  ...///..|
|  ........|
|  ........|
|      .   |
|          |
------------

 Right, four
```

```
------------
|...//+*+//|
|..////////|
|  ////////|
|  ////////|
|  .../////|
|  .../////|
|  ........|
|   ..  .. |
|      .   |
|          |
------------

  Right, five
```

The figure is shown on page 254

Figure D.3: Excitation patterns of the SOFM for different types of motor action vector inputs. The cell with the largest excitation is indicated by "*", "+" means that the excitation of a cell is higher than 0.9, "/" denotes excitations between 0.9 and 0.7 and "." excitations between 0.7 and 0.5.

Two observations can be made from these pictures of the response of the SOFM to different types of input vector:

- The size of the excited area is roughly proportional to the frequency of occurence of the input signal that caused the excitation.
- Related inputs excite neighbouring areas. In this example, we can see that "forward" movements stimulate the central region of the network, "left" movements stimulate the lower region and "right" movements the top region of the net. Within these basic regions there are variations, depending on the duration of the movement.

Topographic mappings such as these are common in biological nervous systems. The striate cortex, for example (one part of the visual system of primates) is organized in topographic fashion, each half of the visual field (of a macaque monkey) is projected onto the striate cortex in a systematic map [Hubel 79]. [Churchland 86] gives a good survey of topographic mappings in neural systems (for topographic mappings of the visual cortex see, for example, [Allman 77]).

Similarly, somatotopic[3] maps can be found in the cortex. Very similar observations to those stated above can be made about such biological systems, ([Churchland 86]):

- Map distortion is based on the population of neurons representing a particular body area. Areas with a higher density of receptors are bigger. [A higher density of receptors will result in a greater number of sensory signals].
- A precise mapping of the body surface onto neurons of the somatosensory cortex is found. Neighbourhood relations of body parts are preserved.

If the nature of input stimuli is altered, for example by joining two fingers of a hand together, the mapping in the somatosensory cortex changes accordingly, see ([Clark *et al.* 88]).

The question that arose for us from these observations was: "Can we use this sort of SOFM to achieve location recognition?" In other words, can

[3]From $\sigma\tilde{\omega}\mu\alpha$, body. What is meant here is that touch, pressure, vibration, temperature and pain sensors are mapped onto the cortex in an orderly fashion.

the mapping of different motor action types onto different regions of the SOFM network, as shown in figure C.3, be used in a system for recognising locations? Our tentative answer is that it seems they can be. In the next sections we explain experiments we devised to do this, the system we finally used and the experimental results of the tests we have carried out so far.

D.4.3 Location Recognition, using one SOFM

Having done these initial experiments with a ten by ten cell SOFM, using the motor action vectors directly as an input to the network, we then tried to combine several of these vectors as input to a network of twelve by twelve cells. The idea was that the response of the network would be correlated with Alder's arrival at a physical location, thus allowing the robot to recognize particular locations. As it turned out, the number of motor action vectors within an input vector that gives best recognition results varies from location to location. Some locations are easy to identify using just a short history (i.e. small numbers of combined motor action vectors), others need a longer history (i.e. a large number of combined motor action vectors). There was no one length of input vector which gives equally good recognition performance with all the corners in the robot's enclosure.

In retrospect, this is not a particularly surprising result. The means by which locations are identified — their "signatures" — are patterns of actions extended over time. As the rate at which motor action commands are issued goes up at corners (obstacle avoidance), the temporal scale of these signatures is dependent upon the distances between corners. In order to be able to identify arbitrary locations it is therefore necessary to use input vectors of different lengths. A fixed size of input vector, whatever it is, will either include extraneous information (complicating the task of the SOFM) or miss possibly crucial signature components (eliminating necessary distinctions).

In some respects, the problem is similar to that encountered in the analysis of natural images in vision: the data is intrinsically multiscalar (though here it is temporal scale that is critical) and so must be subjected to a multiscalar analysis (for this analysis in images see [Marr & Hildreth 80] and [Canny 86]). This motivated our next experiments, in which different sized histories of actions were presented to separate SOFMs in parallel.

D.5 A SOFM Location Recognition System

As in our earlier work on location recognition (see [SAB 91]), the method used is one in which location recognition is based upon the robot recognising its arrival at some location. In other words, location recognition is not based upon recognizing some structural feature of the particular physical location, but on the sequence of actions that have taken place prior to arriving at the particular location. In our earlier work it was the sequence of previous features (corners) that had been detected immediately prior to the current feature (corner) that was used to determine recognition. In this work it is the sequence of motor action commands that have been issued prior to the arrival at the particular physical location that enable Alder to recognize the location. To achieve this we used a system of *seven* independent, two-dimensional SOFMs, working in parallel. Each SOFM consists, as in section C.4.3, of a network of twelve by twelve cells. The input vectors to each of these networks are different, but all are built from motor action vectors as shown in figure C.2. By combining 2, 4, 6, 8, 12, 16, and 24 of these basic motor action vectors we formed seven SOFM input vectors which correspond to increasingly longer histories of the robots motor action changes, (see figure C.4).

The lengths of histories are chosen to cover adequately the expected spectrum of action periodicity. If we think of the sequence of actions generated as the robot circles its enclosure as a pseudo-periodic series, with period roughly equal to the average number of action-vectors generated in a single circuit, then the use of SOFMs tuned to different "frequency bands" allows us to sample the temporal structure of the series across its spectrum and associate these samples with the physical locations whose signatures they are.

In other words, the set of excitation patterns of the seven SOFMs produced as the robot arrives at a particular location (a corner, say) in its enclosure can be used to distinguish this location from all other locations the robot passes through as it wall-follows its way around the enclosure.

D.5.1 The Experiment

The robot was set to wall-follow its way around the enclosure. Every time a new motor action command was issued as a result of the built in wall-following or obstacle avoidance behaviour of the robot, a motor action vector as described in section C.3 was generated. This vector, together with the respective number of previous motor action vectors was presented to each of the seven SOFMs. After a sufficient time, about five times round the enclosure, these feature maps had organized themselves into stable structures corresponding to the topological interrelationships and proportional densities of the input vectors.

After this "learning" period the excitation patterns of all seven networks at a particular location (the *target patterns*) were stored. All subsequent sets of seven excitation patterns generated by new input vectors (*object patterns*) were then compared to the set of seven target patterns. This was done by computing the Euclidean distance (or, alternatively, the city-block distance) between pairs of target and object patterns, see equation C.2 in section C.5.2. If the distance values between each of the seven pairs of

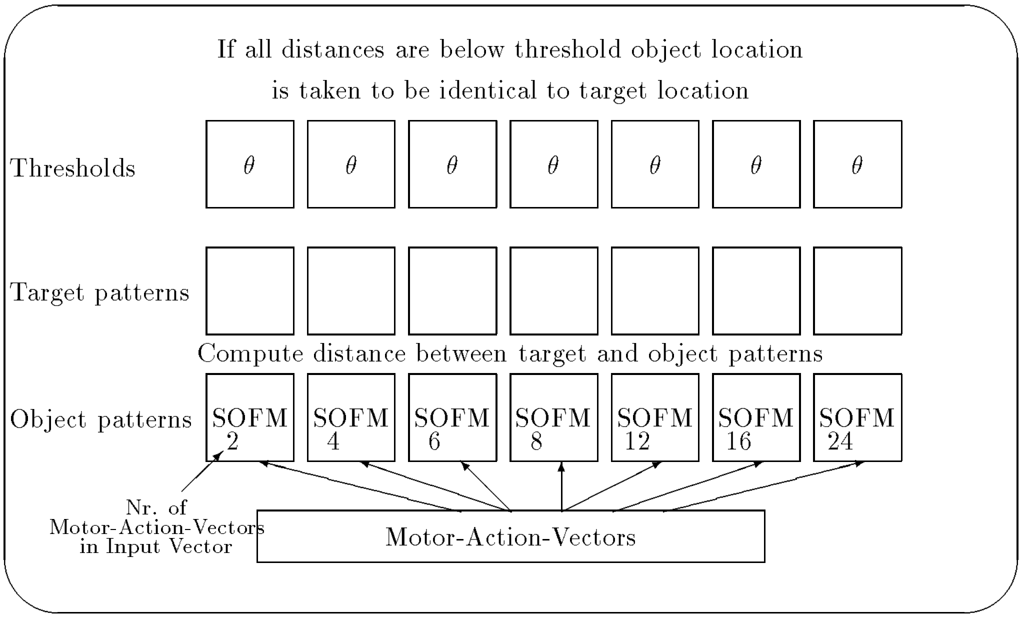

Figure D.4: The System used for Location Recognition

object and target patterns are less than a threshold defined for each pair, the robot is taken to have arrived back at the target location and thus to have "recognized" the previously stored location.

D.5.2 The Mathematics

In mathematical terms, the system works as follows:

1. Compute the output o_{xyj} of each cell at position (x, y) of each network j:

$$o_{xyj} = \vec{w}_{xy} \cdot \vec{\imath}_j \quad j = 1, ..., 7, \tag{D.1}$$

   where $\vec{w}_{xy}$ is the individual weight vector of cell(x, y), and $\vec{\imath}_j$ is the input vector to network j.

2. Compute the distance between target pattern and corresponding object pattern:

   (a) Either the Euclidean distance is chosen,

$$d_j^e = \sum_{x=1}^{12} \sum_{y=1}^{12} (o_{xyj} - o_{xyj}^T)^2, \tag{D.2}$$

   where o_{xyj} is the output value of cell(x, y) of the object pattern j, and o_{xyj}^T is the output value of cell$_{xy}$ in target pattern j.

(b) Or, the city-block distance is computed

$$d_j^{cb} = \sum_{x=1}^{12} \sum_{y=1}^{12} |o_{xyj} - o_{xyj}^T|. \tag{D.3}$$

3. Determine whether object location and target location are identical:

   If $d_j < \Theta_j$ for all j=1 to 7 then object location and target location are taken to be identical. Θ_j is the threshold for network j. For the experiments reported here the Θ_j were chosen by the designer so that patterns obtained at target location and patterns obtained at other locations could be separated. Note that the actual values will, of course, depend on the distance metric chosen. Schemes can be devised that determine these thresholds automatically, but not without supervision.

D.5.3 Results

Data recorded from the robot was used to compute these results, however the actual computation was done off-line on a workstation.

In the experiments conducted so far the robot's task was to identify three particular corners in the enclosure shown in figure C.1: corners H, E and F. To do this the set of seven excitation patterns of the seven SOFMs at the target corners were stored; all subsequent sets of excitation patterns were then compared to these by computing the Euclidean distances between respective pairs of patterns in the set of seven.

Provided a suitable set of thresholds was used (see paragraph 3 in section C.5.2), the robot recognised corner H four times in the subsequent five rounds, and corners E and F in five out of the five times. At no time was a non-target corner erroneously "identified" as a target corner.

D.5.4 Changes in Parameters

Different Metrics

Instead of using the Euclidean distance to estimate similarity between excitation patterns, the "city-block" distance can be used with identical results (see equation C.3 in section C.5.2). City-block distance is computationally cheaper than Euclidean distance.

One-dimensional Networks

Instead of using two-dimensional networks, *one-dimensional networks* can also be used. This considerably reduces the computational cost of the scheme. We used one-dimensional networks of twenty-five cells to test this.

In the case of corners H and E the results were the same as the ones presented above. In the case of corner F, one erroneous identification took place, as well as the five correct ones as before. This seems to indicate that one-dimensional networks can be used instead of two-dimensional ones in this task, except with slightly less reliability. A plausible conjecture for this is that the neighbourhood relationship plays an important role: each cell in a two-dimensional network has eight neighbours, through which excitation can spread, whereas a cell in a one-dimensional network has but two neighbours. The actual number of cells (twelve by twelve as opposed to one by twenty-five) will have some influence, too; but in our experience this is not as important as the neighbourhood relationships.

D.6 Summary and Conclusion

D.6.1 Summary

The robot's task was to recognize particular locations (for example corners) in its world—a simple enclosure. In an earlier publication ([SAB 91]) we showed that this task can be accomplished using self-organising feature maps. The input vector used then contained explicit information about landmarks encountered: that the robot was at a corner, what sort of corner (convex or concave) it was, and information about previous corners encountered.

In the subsequent experiments reported here we tried to reduce the explicit information content in the input vector. We also tried to generate input vectors to the network(s) that contain no direct information about sensor signals. The motivation for the first was to avoid using predefined knowledge, because we believe that predefinition of knowledge limits the robot's flexibility; the reason for the latter was to prove our claim that sensing and acting are in fact closely related.

The input vector we chose thus contained information only about the motor commands of the robot controller, and their duration. Vectors put together from varying numbers of these motor action vectors (2,4,6,8,12,16 and 24) were presented to seven separate self organising feature maps, each two-dimensional of size twelve by twelve cells. In order to identify a particular corner, all seven excitation patterns had to be close enough to a stored set of target patterns. If this was the case, the robot was said to have identified the target location.

The location recognition system performed well in this experiment, recognizing corner H (see figure C.1) in four out of five times, corners E and F in five out of five times, with no erroneous identifications in either case. Using self-organising feature maps to recognize locations adds a high degree of freedom to the robot controller. The robot is able to build its own representations of the environment, independently of the designer.

D.6.2 Conclusion

There are two main conclusions we can draw from the work presented. First, our claim that sensing and acting are closely coupled is confirmed by the success of the robot in recognizing locations based upon the sequence of motor activity which leads to arrival there. The "sensor" being used to provide features on which the recognition is based is actually the behaviour of the robot.

Choosing an input vector that contains no direct information about sensory signals makes the system independent of the actual sensors used. Whether tactile, ultrasonic, infrared or other sensors are used: the location recognition system stays the same.

Second, with this approach the features to be identified by the SOFMs are spread out over time. This contrasts with our early work [Nehmzow *et al.* 89] in which the robot controller learns instantaneous reactions to sensor stimuli. The idea, successfully demonstrated here, of using multiple channels to avoid an assumption of a single natural timescale, merits considerable further investigation since, at first sight, it should be capable of wide application to problems of eliciting temporal structure.

D.6.3 Open Questions and Future Work

We intend to implement a whole system as described above on a mobile robot. This will show us how feasible it is to use this method in applications where real-time processing is important, and where signal noise can bring problems. Implementing such a system on a mobile robot will also enable us to investigate further the questions concerning network size and structure of input vector.

The question of identifying relevant structure over time merits further study and we intend to investigate the limitations of the method used here as well as test alternative approaches.

Acknowledgements

We thank Peter Forster, our colleague on RUR, for his constructive and helpful contributions to this work, and for his comments on earlier versions of this paper.

The work reported here is supported by a grant from the UK Science and Engineering Research Council (grant number GR/F/5852.3). Other facilities were provided by the University of Edinburgh.

Appendix E

Using Motor Actions for Location Recognition[1]

Ulrich Nehmzow
Department of Artificial Intelligence
University of Edinburgh
ulrich@uk.ac.ed.aifh

Tim Smithers
Artificial Intelligence Laboratory
Vrije Universiteit Brussel
tim@be.ac.vub.arti1

[1]Published in Proceedings of 1st European Conference on Artificial Life 1991, MIT Press Cambridge Mass. and London, England, 1991.

E.1 Abstract

We present a Behaviour-based mobile robot that is able to autonomously build internal representations of its environment and use these for location recognition. This is done by a process of self-organisation, no explicit world-model is given. The robot uses an artificial neural network for mapbuilding and additional adaptive processes to achieve successful location recognition. The robot reliably recognises locations in the world and can cope with the noise inherent to the real world.

E.2 Introduction

In an ongoing series of experiments we are using simple mobile robots to investigate mechanisms to support the autonomous acquisition of specified competences. These experiments form a part of our "Really Useful Robots" (RUR) project, the aim of which is to develop adaptive control schemes for autonomous mobile robots which are both flexible and robust with respect to variable and unforeseen situations. So far we have investigated schemes for learning obstacle avoidance, dead-end escape, wall-following and corridor-following competences, [Nehmzow *et al.* 89], learning location recognition using sensor-based feature detectors, [Nehmzow & Smithers 91a], and learning location recognition using motor command sequences, see [Nehmzow *et al.* 91b]. In this paper we present a new location recognition scheme which is not only more robust and reliable, but also simpler than our previous schemes. It again uses information derived from motor commands, rather than directly from sensors, which we believe to be novel in autonomous systems research.

Our previous experiments were done using "Alder", the first of our Really Useful Robots (see page 74). Alder consists of a chassis built from Fischertechnik, an ARC52 controller which uses an INTEL 8052 eight-bit microprocessor and has an on-board BASIC interpreter, an interface card giving independent control for two motors (forward, reverse, and stop for each motor, but with no feedback of distance travelled or angle turned), and up to eight binary sensor inputs. For most of our experiments we have used two whisker sensors which act as omnidirectional tactile sensors. For the obstacle avoidance, dead-end escape, and wall following we used a behaviour-based controller together with a perceptron-like network to provide the necessary plastic element in the learning mechanism (see [Nehmzow *et al.* 89] for more details).

For the location recognition experiments we have been using self-organising networks (see section D.3.1). In our first experiment the self-organising process was fed with input vectors derived from a sensor-based feature detector programmed into the robot, [Nehmzow & Smithers 91a]. The features that were detected and identified were the convex and concave corners of its enclosure (see figure D.2), which it came across as it followed the wall around the internal perimeter. Although this scheme proved to be successful, we were dissatisfied with having to base it upon a hard-wired feature detector—we would prefer not to have to 'tell' the robot so much about its environment. In an attempt to do away with the feature detector we devised a location recognition scheme which uses vectors based upon motor commands. In this scheme an input vector to the self-organising network was constructed each time the motor states changed as a result of a new command. This meant that the number of input vectors produced was significantly larger than in the previous scheme. It also meant that it was harder to derive the information necessary to demonstrate location recognition. In the final scheme we used seven two-dimensional self-organising networks all working in parallel. Essentially this scheme performed a kind of frequency component analysis on the pseudo-periodic sequence of input vectors generated as the robot completed circuits of its enclosure. Although this scheme was successful, and it did remove the need for the explicitly programmed feature detector, we were again dissatisfied with it because of its high complexity and computational cost relative to the previous feature detector-based scheme. We also disliked the fact that the seven thresholds for the seven networks for location recognition had to be set by hand, thus making it dependent on 'magic numbers' which have to be set by the programmer.

This led us to devise a new location recognition scheme which is the subject of the experiments presented here. For this scheme we have used a new robot, 'Cairngorm', the third of the Really Useful Robots[2], shown on page 76.

Cairngorm also uses a chassis built from Fischertechnik, but has a more powerful controller based on a Motorola 68000 CPU and having 128 Kbytes of memory. Its motor control and sensor inputs are similar to that of Alder, but it is programmed in C, rather than BASIC. Otherwise it is essentially the same as Alder.

In the next section we describe the Behaviour-based controller and self-organising scheme used to support the new location recognition scheme. We then describe the experimental setup and procedure adopted; following that, we present our experimental results. We finish with a brief discussion and conclusions.

[2]A much larger mobile robot called 'Ben Hope' is the second in the series and is still under construction.

E.3 Behaviour-based Control and Map-building Process

Cairngorm's control structure consists of three independent behaviours. They are responsible for obstacle avoidance, wall seeking, and mapbuilding, respectively. There is no direct communication between these behaviours except through the world. In combination they lead to successful location recognition behaviour. The mechanisms used in each of these behaviours will now be described in a little more detail.

The *obstacle avoidance* behaviour is a preprogrammed and fixed behaviour that makes the robot turn left for as long as either or both of the two whisker sensors (one on the left and one on the right) are on, i.e., are in contact with an obstacle (the robot is designed to follow a wall on its right hand side). As soon as there are no signals from the whisker sensors the robot resumes moving forwards.

The *wall seeking* behaviour is also a preprogrammed and fixed behaviour. If the robot has not experienced a whisker contact for some preset period of time (about three seconds in our experiments) it makes a right turn (towards the wall). It continues to turn until a signal from the right whisker, or both the right and left whiskers, is received (typically it is just the right-hand-side one), upon which the obstacle avoidance behaviour introduces a left turn away from the wall. The robot then continues to move forward until either the set period of time has elapsed again or it makes contact with an obstacle. In this way, the robot is able to both avoid obstacles, follow walls, and negotiate the corners of its enclosure without these being explicitly specified tasks achieved using explicitly programmed strategies. In this sense they are examples of emergent functionality as defined by [Steels 91].

In the experiment presented here both the obstacle avoidance and wall following behaviours were preprogrammed. However, we have previously shown that our robots can acquire this kind of obstacle avoidance and wall following competences through learning from interaction with their environments (see [Nehmzow *et al.* 89] for details).

The third behaviour is *mapbuilding*. By 'mapbuilding' we mean something more like *taking notes* of particular experiences, rather than constructing a geographical map or floor-plan. For this a self-organising network (see [Kohonen 88]) is used. It is constructed ('trained') using input vectors derived from the motor action commands produced by the obstacle avoidance and wall-seeking behaviours. We next briefly review the details of this kind of network before describing in more detail the location recognition behaviour.

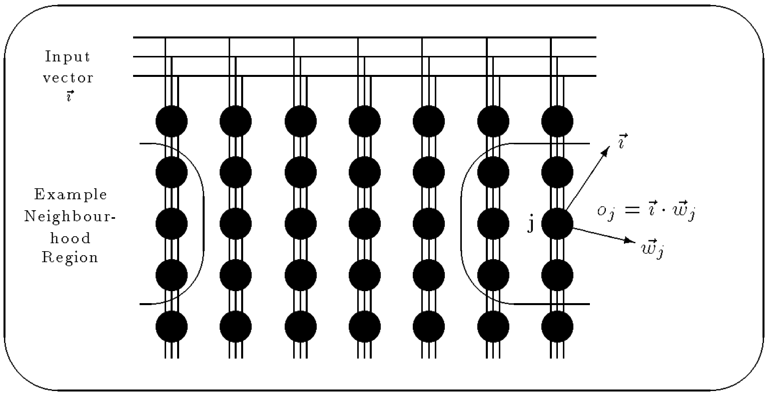

Figure E.1: A two-dimensional self-organising network.

E.3.1 Self-Organising Networks

Consider the two-dimensional self-organising network (SON) given in figure D.1.

Each cell j of the SON has an individual weight vector $\vec{w}_j$ of unit length. Each normalised input vector, $\vec{\imath}$, is fed to all cells[3]. The output o_j of cell j is determined by the dot product of input vector $\vec{\imath}$ and the weight vector $\vec{w}_j$ of cell j:

$$o_j = \vec{w}_j \cdot \vec{\imath} \tag{E.1}$$

The cell with the strongest response, the largest output value, is selected. This cell as well as all neighbouring cells within a defined neighbourhood region are then modified according to the following equation:

$$\vec{w}_j(t+1) = \vec{w}_j(t) + \eta(\vec{\imath} - \vec{w}_j(t)) \tag{E.2}$$

where η is the so-called 'gain'—a value that determines the amount of change (it is typically set at 0.2). Weight vectors outside the specified neighbourhood[4] remain unchanged. After several 'epochs', i.e. presentations of input vectors to the network, the net develops regions which respond most strongly to particular types of input vectors. In this way a mapping is developed whereby different input vectors are mapped onto different regions of the network (always an injection, often a bijection).

[3]For self-organisation, normalising the input vector is not strictly necessary. However, for numerical comparison of two responses of the network, which we do, it is.

[4]The neighbourhood size is often chosen to be decreasing over time, so that self-organisation occurs over larger areas of the network early on and then becomes more local, but in our case we keep the neighbourhood region fixed.

E.3.2 Location Recognition Behaviour

On Cairngorm we have used a one-dimensional, ring-shaped SON of 50 cells, as in earlier experiments with Alder, with a neighbourhood region of one node either side. As the robot makes its way around its enclosure, wall-following and avoiding obstacles, a moving average of durations of turn actions is computed. If a turn action occurs which takes longer than the current average (a 'significant turn action') an input vector is constructed and fed into the self-organising network and the modification to the weight vectors of the net as defined above is performed. After about two to three times round the enclosure the one-dimensional ring develops a stable enough structure to be used to recognise particular locations. This is demonstrated by instructing the robot (by pressing a microswitch mounted on it) to store the excitation pattern of the self-organising ring when the robot is at the location that is to be recognised ('home' location). This stored pattern is then compared with all subsequent excitation patterns. If the pattern at the current location of the robot is found to be sufficiently similar to the stored one, the robot indicates that it has arrived back at the home location. The comparison is computed using the 'city-block' distance ϵ between two excitation patterns, and is given by:

$$\epsilon = \sum_{k=1}^{50} |o_{s_k} - o_{c_k}|, \tag{E.3}$$

where $\vec{o}_s$ is the stored excitation pattern, and $\vec{o}_c$ is the current excitation pattern. By using the city-block distance measure (or Euclidean distance measure, for that matter) we are effectively performing a vector quantisation, so the property of topology-preserving mapping of SON is not exploited. The advantage of this is that the network can be used for location recognition even in the early learning stages. However, once the net is well settled the index information of the most excited cell alone should suffice to recognize locations.

E.4 The Experimental Results

In this section we present our experimental results. We begin by describing the experimental setup.

E.4.1 Experimental Setup

The robot is placed in a simple enclosure, containing rightangled convex and concave corners as well as straight walls. Figure D.2 shows the layout of the enclosure. The letters indicate locations where a turn action usually exceeds the average duration and therefore where an input vector to the SON is usually generated.

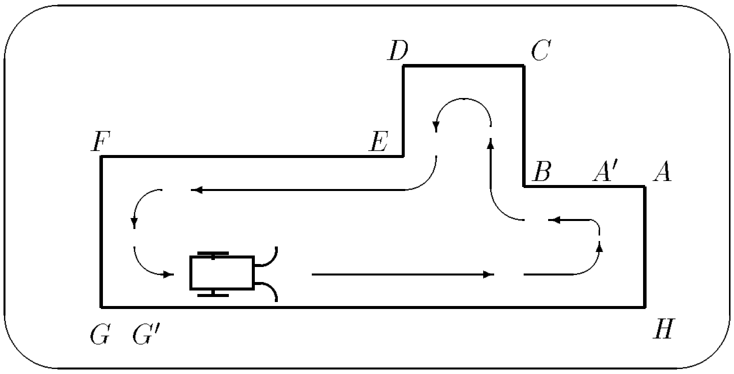

Figure E.2: The experimental enclosure for location recognition.

Once started, the robot follows the wall on its right hand side round the enclosure. This is achieved, as we mentioned before, by a combination of the obstacle avoidance and wall seeking behaviours. The mapbuilding behaviour monitors the turn actions commanded by the other two behaviours. Whenever a turn action takes longer than the current average turn action time, an input vector (see input vector 1, in figure D.3), is generated and presented to the SON. The response of each cell in the ring is then determined and the relevant weight vectors are updated, as described in section D.3.1. The robot is left to wall follow its way around the enclosure for typically two to three rounds, then a "home" corner is selected by pressing the microswitch mounted on the robot. This results in the current excitation pattern being stored in memory. From this point on all subsequent excitation patterns of the SON are compared to the stored pattern. If the city block distance between the goal-pattern and the current excitation patterns is less than $\frac{1}{3}$ of the average city-block distance, the robot assumes it is back at the goal location and a red light (mounted on the interface board) lights up.

As can be seen from input vector 1 in figure D.3, the information used as input to the self-organising network consists of the type of the current turn action (right or left), and the types of the previous two turn actions, together with the elapsed time between each pair.

In order to investigate the performance of this location recognition scheme we used two other input vectors, see input vector 2 and 3 in figure D.3, each one containing less information than the previous one. Input vector 2 consists of the types of the current and previous turn actions and the elapsed time between them, and input vector 3 consists of just the types of the current and previous turn actions. The experimental setup when using each of these three kinds of input vector was in all other respects the same.

The results obtained for all three input vectors are presented in the next section.

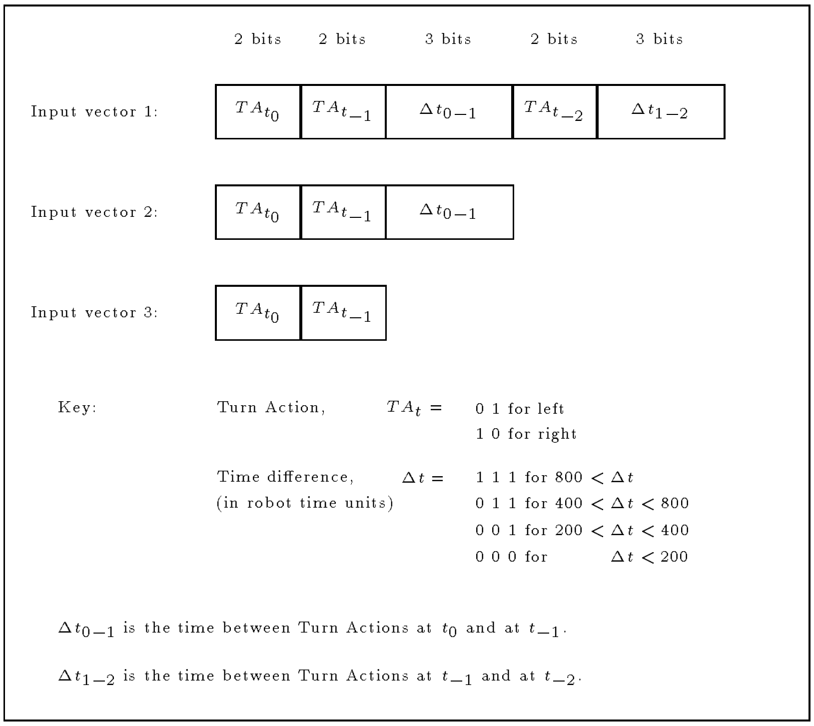

Figure E.3: Input vector definition for the Self-Organising Controller.

E.4.2 Experimental Results

Table 1 shows the location recognition results obtained using input vector 1 (see figure D.3).

Location	Recognitions	Total no. of Visits
H	3	4
B	3	3
C	3	3
D	3	3
E	3	3
F	3	3
G	3	3
G'	2	4
A	3	3
A'	3	3

Table 1: Location recognition using input vector 1.

The letters refer to the locations shown in figure D.2. Except for A' and G', they correspond to the corners of the enclosure. This is because turn actions that take longer than average are produced at these places. The locations A' and G' are produced because the robot typically takes two long turn actions to negotiate these particular corners.

The one 'missed' recognition at corner H occurred early on in the run and was due to the network not having settled down enough for a successful recognition to be registered. The two 'misses' at G' occurred because on these occasions the robot got round corner G with one turn action, and so the region of the SON corresponding to an input vector having G', G, and F was not excited by any input vector.

	A	A'	B	C	D	E	F	G	G'	H
A										
A'									X	
B										
C										
D										
E										
F										
G										
G'		X								
H										

Table 2: Confused locations when using input vector 1.

Table 2 indicates the pairs of locations which were confused during the run.

In this case it was only locations A' and G' which were confused. This occurs because the input vectors at these locations (built from H, A, A' and F, G, G' respectively) are similar, and thus excite the self-organising network in the same way.

Table 3 presents the location recognition results when using input vector 2. As can be seen, there is a degradation in performance with 'missed' locations occurring at D, F, and G. The failure to recognise location F here was due to an input vector not being generated at location E, thus leading to an 'odd' vector at F. This occurred only once and was the result of the inevitable variation in actual behaviour experienced when using real robots (even simple ones) in a real environment. This was, however, a one-off event and never observed to occur again.

Location	Recognitions	Total no. of Visits
H	3	3
B	3	3
C	3	3
D	3	4
E	3	3
F	3	4
G	4	5
G'	3	3
A	3	3
A'	3	3

Table 3: Location recognition using input vector 2.

	A	A'	B	C	D	E	F	G	G'	H
A		X			(X)			X	X	
A'	X							X	X	
B										
C										
D										
E										
F										
G	X	X			(X)				X	
G'	X	X						X		
H										

Table 4: Confused locations when using input vector 2.

The degradation in performance can be seen more clearly in table 4, which presents the pairs of confused locations. The bracketed pairs denote occasional confusions.

When we reduce the amount of information in the input vector still further, and use input vector 3, we obtain the results presented in tables 5 and 6. Here we can see that the robot is still able to recognise locations, but its ability to distinguish certain pairs of them is significantly diminished.

Location	Recognitions	Total no. of Visits
B	3	3
C	3	3
D	2	2
E	2	2
F	2	2
G	2	2

Table 5: Location recognition using input vector 3.

	A	A'	B	C	D	E	F	G	G'	H
A		X			X			X	X	X
A'	X				X			X	X	X
B						X				
C							X			
D	X	X						X	X	X
E			X							
F				X						
G	X	X							X	X
G'	X	X						X		X
H	X	X						X	X	

Table 6: Confused locations when using input vector 3.

As we would expect, the performance of the whole system depends on the information put into the self-orgainsing network. If the input vector contains insufficient information, then reliable location recognition is impossible. From the results presented above we can see that reducing the information content of the input vector does not affect the robot's ability to recognise a non-wall type environmental feature (corners in this case), but it does affect its ability to differentiate some pairs of such locations. The less informative the input vector, the more locations "look the same" to the robot.

E.5 Discussion

E.5.1 Discovering Significant Environment Structure

In our previous location recognition scheme based upon motor actions we required a set of seven self-organising two-dimensional networks to get the robot to reliably recognise (and distinguish between) locations. Yet in the scheme we presented above we only need one one-dimensional network (ring) to achieve similar performance results. We can explain this effect by observing that our new scheme works by filtering the sequence of motor commands so that only those *not* generated by following a straight wall are used to build input vectors to the self-organising structure, whereas in the previous scheme the set of seven networks had to do this filtering work implicitly.

Another way of viewing this is to say that the sequence of motor commands produced as the robot wall follows its way around the enclosure contains two kinds of structure. One kind occurs at a high frequency and is produced by the wall-following actions. The other kind has a lower frequency and is produced by the corners, or non-straight-wall features of the enclosure. It is this second type of structure in the motor action commands that contains information about significant structure in the robot's environment (corners in this case); the first type merely reflects the fact that the robot's environment has straight walls in it—a rather less useful piece of information for location recognition.

E.5.2 Setting Thresholds

When we first started to experiment with the scheme presented above, the turn action time threshold, used to distinguish wall-following actions from other actions, was set by hand. We arrived at the particular threshold value by carefully observing the robot's behaviour and choosing it such that it would differentiate between motor actions performed at a 'significant' location (usually a corner) and those performed elsewhere (while wall following). Later we implemented the moving average calculation, thus removing the need for this 'magic number' to be set by hand (by doing this we have effectively introduced two other values to be set: the time window over which the average duration of motor actions is computed and the proportion by which the duration of a motor action has to exceed this average to become a "significant action". However, both these values are not very critical and far easier to set than the previously needed thresholds). Similar mechanisms are found in biology. Pigeons, for example, extract the changes in air pressure generated by changes in altitude of a few feet by ignoring the total strength and only measuring differences around some mean, [Gould 82]. Using this simple device in our robot means that it does not need any predefined knowledge about thresholds and significant motor actions, it finds this out for itself. This approach has further advantages: The robot is able to adjust its assessment if the world or the robot change. Cairngorm's moves, for example, become slower with decreasing battery charge. If a fixed threshold for determining significant moves was used, the performance of the robot would change, simply because motor actions take longer. However, because the average turn action will also take longer, Cairngorm is able to adjust for this and so maintain its performance.

A similar threshold value is used in the comparison between the stored ('home') excitation pattern and all subsequent patterns. Once again, we started by determining the required value for this empirically and setting the value in the program by hand. But having devised the successful averaging mechanisms to set the turn action time threshold we decided to try a similar device for the comparison threshold. The robot thus computes a moving average of all the city-block distances between the stored and current excitation patterns. Once a distance is smaller than $\frac{1}{3}$ of this average distance, the robot indicates that it recognises the current pattern as being the same as the stored pattern, and thus that it recognises its current location as the 'home' location.

E.5.3 Why 'Motor Actions' and not 'Sensor Signals'?

Sensing and acting are so tightly coupled that it seems fair to say that they are two sides of the same coin. In earlier experiments we have used schemes based on sensory information to recognise locations (see section D.2). In the experiments reported here we have obtained the same results as in the earlier work, using motor information rather than sensor information. This was done to prove exactly this point: that sensing and acting are tightly coupled.

There can be practical advantages in using motor action information: the amount of information to be processed is often smaller because the robot is effectively acting as an analog computing device. Also (and perhaps more importantly for practical applications), the controller becomes independent of the actual sensors used. If necessary, sensors can be replaced without affecting the actual controller.

E.5.4 Comparison with Animal Navigation

There are numbers of well studied examples of navigation by learned location recognition in the biological literature. In an exploratory phase bees store *visual images*, which are then associated with a motion vector that gives the direction towards the hive, see ([Waterman 1989], [Gould & Gould 88] and [Cartwright & Collett 83]). When released some distance away from the hive, bees can find their way back using these acquired visual images by comparing them with the current images and calculating the appropriate flight direction to take them to their hive, [Collett 87]. In a similar way our robot stores "motor action images" and later compares stored images with current ones in order to recognise locations it has been at before. However, no vectors pointing homewards are associated with these images.

Other kinds of sensor-based navigation behaviour schemes have also been documented. For example, pigeons and salmon navigate using *olfactory information* ([Gould 82], [Hasler *et al.* 78]), and it seems possible that pigeons also use *acoustic (infrasound) and magnetic sensors* to find their way home [Gould 82]. Bees and ants also use the *polarisation of the light* of the sky to navigate [Wehner 76].

All these biological examples appear to just use sensor information and they are clearly reliable and robust. However, our experiments with simple robots have shown that motor-action based schemes can be used as well in order to achieve high degrees of reliability and robustness. The reasons for this apparent difference are not clear to us.

E.6 Summary and Future Work

In earlier work we demonstrated location recognition schemes using sensor-based feature detectors and using motor action changes. In this paper we present a new location recognition scheme, which again uses information about motor actions, but which is simpler and yet equally reliable and robust. As a part of this new scheme we have also incorporated automatic adaptive threshold setting mechanisms to reduce the number of parameters ('magic numbers') which have to be set by hand.

In our ongoing series of experiments to investigate the autonomous acquisition of specified competences we have demonstrated in separate schemes the acquisition of obstacle avoidance, wall-following, corridor following and location recognition competence by the robot. Our plan for the future is to put all these schemes together in one robot to demonstrate the staged learning of all the competences required for simple navigation in an autonomous robot.

Acknowledgements

The work reported here is supported by a grant from the UK Science and Engineering Research Council (grant number GR/F/5852.3). Other facilities were provided by the Department of Artificial Intelligence at Edinburgh

University. We would like to thank John Hallam for the stimulating discussion about this work. We thank Claudia Alsdorf, Peter Forster and Barbara Webb for reading earlier versions of this paper and their helpful comments. Thanks also to the staff of the mechanical, electronic, and photographic workshops of the Department of Artificial Intelligence for their help and support.

Appendix F

Learning Multiple Competences: Some Initial Experiments[1]

Ulrich Nehmzow
Department of Artificial Intelligence
University of Edinburgh
ulrich@uk.ac.ed.aifh

Tim Smithers
Artificial Intelligence Laboratory
Vrije Universiteit Brussel
tim@arti1.vub.ac.be

[1]To be presented at workshop on "Neural Networks and a New AI", August 3rd, 1992, as part of the European Conference on Artificial Intelligence, Vienna 1992

[2]Names appear in alphabetical order, with both being principal authors on this occasion. Tim Smithers currently holds the SWIFT AI Chair in the VUB AI Laboratory.

F.1 Abstract

The vast majority of work on learning in artificial intelligence has focussed upon techniques for developing a single competence. Autonomous systems, however, have to possess a number of competences in order to operate effectively. How learning of multiple competences is to be achieved in autonomous robots is thus of fundamental importance to any understanding of learned autonomous behaviour which involves more than one competence. In this paper we present some initial experiments to investigate the staged learning of multiple competences in an autonomous mobile robot. This work combines our previous work on the separate learning of locomotion competence and a simple navigation competence.

F.2 Introduction

In 1949 Donald Hebb wrote:

> "All learning tends to utilize and build on any earlier learning, instead of replacing it, so that much early learning tends to be permanent ... the learning of the mature animal owes its efficiency to the slow and inefficient learning that goes on before, but may also be limited and canalized by it." — [Hebb 49].

Despite Hebb's widely acknowledged contribution to modern connectionist learning research, the vast majority of work on learning in artificial intelligence (AI) has focussed upon techniques for developing a single competence. This is true of research in both symbol processing machine learning and connectionist learning. Autonomous systems, however, generally have to possess a number of competences in order to operate effectively. In the case of an autonomous mobile robot, for example, it must be able to get around while avoiding obstacles and not getting stuck, and it must also be able to navigate in its environment, to recognize and find its way to and from locations. How learning of multiple competences is to be achieved in autonomous robots is thus of fundamental importance to any understanding of learned autonomous behaviour which involves more than one competence, just as it is in understanding the biological examples Hebb was writing about in 1949.

The sequential development of multiple competences in humans and primates has received attention in both comparative psychology and the neurosciences. McGonigle and Chalmers ([McGonigle & Chalmers 77] and [McGonigle & Chalmers 92]) working in Edinburgh on a comparative analysis of the staged development of non-linguistic skills in children and monkeys, have identified comparable staged sequences of skill development. Thatcher *et al.*, [Thatcher *et al.* 87], using EEG (electroencephalogram) phase and coherence measures, have offered evidence for sequential periods of rapid growth and consolidation in the development of cortico-cortico connections between hemispheres in human brain development. An important conclusion of this psychological and neurological work is that the sequential development of multiple competences is the result of genetically determined changes in the nervous system and not just a consequence of antecedent experience: their development is a matter of 'design' not chance, [McGonigle 91].

A related question concerns the modularity of the underlying control system and its architectural organization. Again, work in the neurosciences offers strong evidence for the decomposition of biological nervous systems into psychobiological subsystems working concurrently, each dedicated to supporting particular competences. For example, clinical studies of amnesia in humans point to the existence of distinct memory systems—procedural memory and episodic memory systems. The idea of multiple memory systems is in marked contrast to the single monolithic idea of memory function found, almost universally, in AI research, both symbol processing and connectionist. The partiality of memory degradation found in the clinical studies suggests that (human) memory is a collection of subsystems each having evolved to meet specific requirements. As Sherry and Schacter ([Sherry & Schacter 87]) argue, these requirements collectively may be functionally incompatible thus necessitating separate memory systems to satisfy them.

Contrary to recent trends (see [Brooks 86] and [Beer 90], for example) we believe that these results from human and primate cognitive neuroscience research offer pointers to important design principles for autonomous robots. While the investigation of legged locomotion through robot construction and simulation, as pursued by Brooks and Beer, is in itself interesting, and probably of engineering import, we do not believe it can lead to the identification of strong constraints on the processes and mechanisms necessary for intelligent behaviour. We do, however, believe in a 'bottom-up' approach to the construction and testing of controllers for autonomous robots and the modularity of single competence supporting subsystems, as advocated by Brooks.

In this paper we present some initial experiments to investigate the staged learning of multiple competences in a simple autonomous mobile robot. This work combines our previous work on learning a locomotion competence and a simple navigation competence. In the next section we briefly review the control architectures and connectionist techniques used in this previous work. In section E.4 we describe the controller used to support the sequential learning of multiple competences and the results obtained so far. We then discuss some related work on learning in autonomous robots and conclude with some final comments and further research directions.

F.3 Background: Single Competence Learning

In an ongoing series of experiments we have been using simple autonomous mobile robots to implement and test mechanisms to support the autonomous development of specific (single) competences. The aim of this research is to investigate adaptive control schemes for autonomous mobile robots that are both flexible and robust with respect to variable and unforseen situations. Here, unforseen situations are those that have never before been experienced by the robot nor anticipated (and so catered for) during its design.

So far we have implemented and tested control schemes for learning a locomotion competence — for obstacle avoidance, dead end escape, wall-following, and corridor-following, [Nehmzow *et al.* 89] and [Daskalakis 91], and learning a simple navigation competence using sensor-based feature detectors, [Nehmzow & Smithers 91a], and, alternatively, motor command sequences, [Nehmzow *et al.* 91b] and [Nehmzow & Smithers 91b]. Throughout these experiments we have used self-organizing processes implemented using connectionist computing techniques (see below for more details). The robots used have onboard microprocessors connected to two drive motors and simple (binary) sensors, see [Nehmzow & Smithers 91b] and [Donnett & Smithers 91] for details.

F.3.1 Learning Locomotion Competences

The self-organizing controller used for learning the locomotion competence is based upon a perceptron network used as an associative memory together with a performance monitor, and an action selector, see figure E.3.1.

The associative memory is used to relate input signals derived from sensors to motor command signals which produce forward motion, reverse motion, and left and right rotation motions. The monitor compares the current motor and sensor signals with a set of *instinct-rules* which define the internal conditions that the controller continuously tries to satisfy. The action selector selects which motor action to actually perform at any time.

If the monitor detects that an instinct-rule is not satisfied the current input vector (derived from the sensor signals) is applied to the associative memory which then produces an ordering of the four possible motor actions it currently associates with the particular input vector. The action selector then picks the first action in the list and it is executed for a fixed period of time. If after this action the violated instinct-rule is now satisfied the current association is confirmed by 'teaching' the perceptron network with the input vector motor action pair. If, on the other hand, the instinct-rule is still not satisfied the action selector picks the next action in the ordered list and executes it. This action is executed for a longer time in an attempt to cancel the effect of the previous (unsuccessful) motor action. This pattern

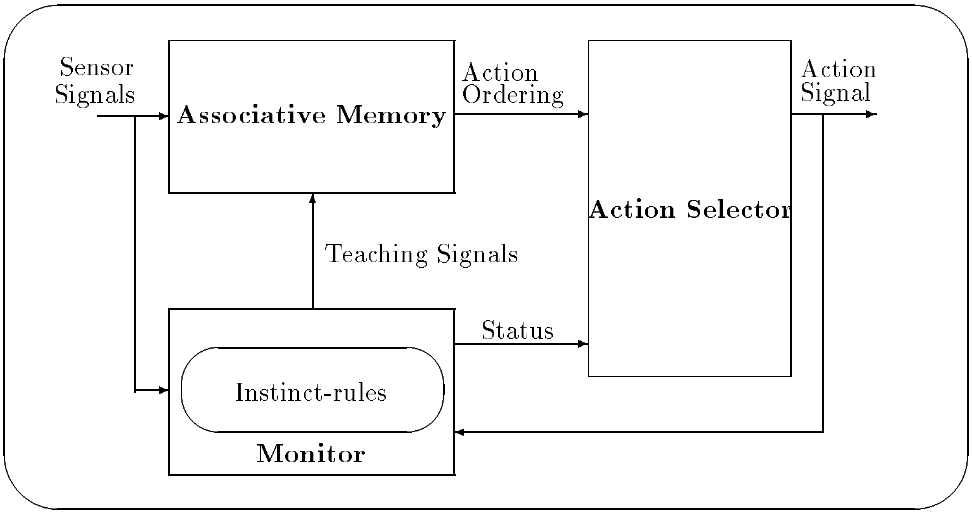

Figure F.1: Self-organizing controller for locomotion competence learning

of activity is repeated until an action is executed which does result in the violated instinct-rule being satisfied. This successful motor action is then taught to the associative memory, thus modifying the controller in a way that is intended to result in a fast restoration of the violated instinct-rule the next time it occurs.

In this way the robot is able to both learn and to subsequently adapt sensory-motor relationships that enable it to achieve various tasks encountered during locomotion. The advantage of this type of self-organizing controller, over more conventional schemes, is that it offers greater flexibility in the face of variable and unforseen situations. If the situation changes so that its current sensory-motor associations become maladapted it is able to adjust itself until competent performance is restored. This controller can even deal with such radical changes as swapping the sensors over from one side of the robot to the other. At first, it performs the wrong actions, but it soons adjusts the mapping to deal with the new situation.

Obstacle Avoidance	**Wall Following**	**Corridor Following**
• move forward • avoid contact	• move forward • avoid contact • make contact after a while	• move forward • avoid contact • make contact after a while • make contact on alternate sides

Table 1: Instinct-rule sets for different locomotion competence learning

The sets of instinct-rules used for learning obstacle avoidance and dead end escape, wall following, and corridor following are presented in table 1. As can be seen from these three sets, the wall following and corridor following competences are achieved by the simple addition of extra instinct-rules. No other modifications to the controller are necessary. It should also be noted that each set contains mutually incompatible rules. It is the interactions which are produced by these incompatibilities that give rise to the learning of the desired locomotion competences. Another aspect of this controller is that it depends upon the set of motor actions used being sufficiently orthogonal in the sense that forward and reverse actions produce no rotations and can be used to cancel the effects of the other, and that left and right rotations produce no forward or reverse motion and can again be used to cancel each other. This is achieved by careful design of the physical configuration of the robot and by keeping the speed of the robot low. Daskalakis, [Daskalakis 91], reports on what can be done when these conditions are not sufficiently satisfied.

F.3.2 Learning Simple Navigation Competences

In a second series of experiments we have been investigating the use of self-organizing maps based upon the techniques of Willshaw and von der Malsburg, [Willshaw & v.d.Malsburg 76], and Kohonen, [Kohonen 88] to support the learning of simple navigation competences.

For these experiments the robot was given (rather than expected to learn) a wall following competence. It was then placed in an enclosure, built of straight walls and rightangle corners, and left to find its way around the perimeter. Several different schemes have been investigated for supporting map learning in this robot. First we used a hard wired feature detector to enable the robot to detect the convex and concave corners in its environment as it moved around the walls of the enclosure. As each corner was detected, information about its type (concave or convex) and the distance from the previous corner and its type, was used to constructed an input vector for a one-dimensional self-organizing network. After several times round the enclosure the robot was able to use this network to recognize when it had returned to a nominated location. By extending the input vector to include in formation about the previous two and three corners we were able to improve the accuracy of the location recognition until it was perfect (in the experimental enclosure used), [Nehmzow & Smithers 91a].

In an attempt to move away from this rather inflexible feature detector-based scheme, we implemented a scheme which used motor commands as the basis for the input vectors to the self-organizing network. In this version an input vector to the self-organizing network was constructed each time the motor state changed as a result of a new motor command. This meant that the number of input vectors generated increased significantly over the previous scheme. It also meant that it was harder to derive the information required to achieve reliable location recognition. In its final form we used seven two-dimensional self-organizing networks all working in parallel. Essentially this scheme performed a kind of frequency component analysis on the pseudo-periodic input vector sequence generated as the robot wall follows its way around its enclosure, [Nehmzow *et al.* 91b].

Although this scheme was successful, and it did remove the need for an explicitly programmed feature detector, we were dissatisfied with its large computational cost relative to the first scheme. We also disliked the need to have to set seven different threshold values (magic numbers) by hand in order to get the scheme to work well. This led us to devise a third map-building scheme which again used no hardwired feature detectors but which was computationally much cheaper than the second scheme. In this third map-building system the robots calculates a moving average of the duration of the turn actions it executes to make contact with the wall as part of its wall following behaviour. Then, if a turn action occurs that is significantly different from this average, (a significant turn action) an input vector is generated using information about the type of turn (left or right) and the time since the last significant turn action, [Nehmzow & Smithers 91b]. This scheme proved to be as reliable as our first scheme and only a little more expensive in terms of computation. The use of the moving average calculation also proved to be an effective way of implementing the required novelty filter since it was robust with respect to the change in the average time taken by the robot to make turns of the same magnitude as the batteries drained.

F.4 Experiments in Multiple Competence Learning

Having implemented and tested self-organizing controllers to support the learning of locomotion and simple navigation competences separately, we were in a position to attempt to put them together to form a controller which could support the learning of both competences. This attempt to combine two learning schemes raises the question of how we prevent them from interfering with each other. In other words, how is the controller to structure the combined learning of these two competences? It was clear from our previous experiments that we could not leave the robot to learning everything at the same time, and expect everything to sort itself out automatically. What is required is some predetermined control of the staging of the two types of learning.

F.4.1 Corridor-Following and Maze Learning

Before attempting to combine learning of the locomotion and navigation competences, we decided to carry out a preliminary experiment in which to

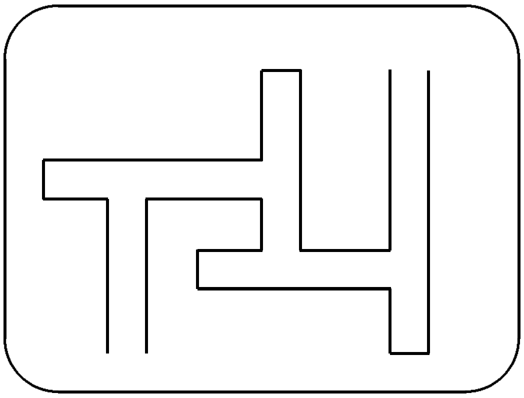

Figure F.2: A typical simple maze used in the corridor-following and maze learning experiments

test a staged learning controller. To do this we built a controller that could learn corridor following and which would then use this competence to detect junctions in a simple maze consisting solely of T-junctions, see figure E.2.

To make the experiment more interesting, the robot not only has to recognise junctions in the maze, but also the dead ends. In addition we also wanted the robot to remember which way it has to turn in order to take the most direct route through the maze a second time. It did this by making a random choice of direction to turn in every time it detected a junction, then if it next detected a dead end it knew the choice was wrong and it had to backtrack to the previous junction, but if it detected a junction next it knew that the previous choice was correct, and this was recorded in a list of turns to make at each junction.

The junction and dead end detection was achieved in a way similar to the corner detection mechanism used in our third map-building scheme. If, during corridor following a turn was made that took significantly longer than the current average turn time needed to make contact with the wall, a junction was taken to have been detected. If the time taken to move way from contact with the wall took significantly longer than the current average time for obstacle avoidance, then a dead end was taken to have been detected.

Now for this scheme to work, the learning of the obstacle avoidance and corridor following competences must be completed and representative average times for obstacle avoidance and wall seeking established. If this were not the case, the self-adapting modifications made by the controller to try to establish the appropriate sensory-motor relationships for effective locomotion (forward motion with obstacle avoidance and corridor following in this case) would interfere with the junction and dead end detection strategies. It is therefore necessary, in this setup, to terminate further learning by the locomotion controller before maze learning starts.

The following simple strategy was used to decide when to terminate this first stage of learning. In order to have developed an effective locomotion competence the robot needs to have encountered every possible input signal configuration at least once, but preferably more than once. The input vector used in the corridor following version of out controller contains four bits[3] which means that the number of different possible configurations is sixteen. In practice there are, however, less configurations since a condition in which contact on the left and right is hardly ever made in corridor following. Also, because, given the way the controller is set up, it is not possible for no contact to be represented in an input vector, the number of possible configurations is eight. Locomotion competence learning was therefore terminated if the robot has only selected motor actions that directly restore instinct-rules as they become violated eight times in a row. When this occurs no further modifications are made to the associative memory of the locomotion controller, and maze learning begins.

In a series of runs the robot was placed at the beginning of the first corridor of the maze and allowed to learn the required locomotion competence and to then find its way through the maze. It was then replaced at the beginning to see it would then take the most direct route through the maze, that it should have discovered the first time. On all occasions that the locomotion learning had taken place and been terminated before the first junction maze learning proved to be successful. Only on those occasions that locomotion learning had not been completed before the first junction did the robot run into trouble, sometimes so much so that maze learning did not take place (at least not in the length of time we were prepared to wait for it to do so). From these results we concluded that the simple staged learning controller used was suitable for application in an attempt to learn locomotion and navigation competences.

F.4.2 Wall-Following and Mapbuilding

In this experiment the robot was equipped with a controller formed from a combination of the locomotion competence learning controller described in section E.3.1, the the third version of our navigation learning controller described in section E.3.2, and the stage controller tested in the preliminary experiment described above.

For this experiment an enclosure similar to that used in the previous navigation learning experiments was used. It was formed of straight wall sections and rightangle concave and convex corners. In addition a removable barrier was installed to form a simple rectangular subenclosure. The

[3]The four bits are used to represent left contact or no left contact, right contact or not right contact, plus four bits representing the same information but for the previous input time.

robot was then placed in the subenclosure and allowed to learn the required obstacle avoidance and wall-following locomotion competence. When this had been completed, according to a similar criterion of that used above, the robot indicated (with a small lamp mounted on the robot) that the locomotion controller had been fixed. The barrier was then removed and the robot allowed to wall follow its way around the full enclosure. After about three complete circuits of the full enclosure the self-organizing map used stabilises sufficiently for location recognition to then successfully take place.

In a series of repeated experiments the robot was able to learn the require locomotion competence in the rectangular subenclosure and to then successfully support effective mapbuilding behaviour for its navigation learning phase. We were therefore able to demonstrate successful structured learning of locomotion competence learning and simple navigation competence learning, albeit for a simple robot in a simple environment, using a combination of the controllers developed to support separate competence learning together with a simple learning staging controller.

F.5 Related Work

The use of connectionist learning techniques in control is now a well established research area, see [Miller *et al.* 90] for a survey of recent work. Much of this work is conducted only as computer based simulations[4], see [Sutton 91], [Long-Ji 91], [Prescott & Mayhew], and [Tyrrell & Mayhew 91], for example, and are all aimed at learning what we call single competences. Another important difference between the learning controllers we describe here and those based upon the reinforcement learning techniques typically used is that our controllers learn much much faster in real environments. Kaelbling, [Kaelbling 91], represents an important exception in that she has also tested her algorithms on a real robot, as have Mahadevan and Connell, [Mahadevan & Connell 91], who have used reinforcement learning techniques in a subsumption type architecture (see below). Though their robots learns several behaviours it still only learns one competence in our terms.

The 'layered' structure of our multiple competence learning controller is similar in style to Brooks' Behaviour-based subsumption architecture, [Brooks 85] and [Brooks 86], though different in implementational detail. In particular, we do not decompose task achieving competences into separate behaviours and nor are the two parts of our controller so tightly integrated as the individual behaviours typically are in a subsumption architecture.

A related mapbuilding scheme developed for a mobile robot is that of Mataric, [Mataric 91]. This scheme, which is claimed to be modelled after

[4]They are called simulations, though often they cannot fairly be described as such since no attempts are made to validate them against the real world behaviour they are presumed to model.

the mapping function of the rat hippocampus, is similarly computationally simple and reliable in practice. It is also based upon the subsumption architecture of Brooks. It differs from our scheme in that it works by identifying physical locations in terms of recognising features of the location, as opposed to recognising the arriving at locations, as is the case in our navigation scheme.

Though related in the sense that it either uses reinforcement type learning schemes, connectionist techniques, or similar mapbuilding techniques, none of the above work deals with the problem of learning multiple competences. The question of how any of these other systems might be combined into larger systems that do support multiple competence learning thus remains an open question.

Finally, and on a slightly different theme, the work we present here is closely related to some ideas presented by Clarke in a somewhat neglected, but important, paper called "Being there: why implementation matters to cognitive science", [Clark 87]. In this paper Clarke argues cogently that "... action and cognition may need to be treated as inextricably inter-related parts of complex systems which understand their environment." In doing so, he questions the almost universal adoption by cognitive science, and we can say by artificial intelligence (AI) as well, of the philosophical doctrine of *machine functionalism*. This asserts that the study of the mind can be safely disconnected from any consideration of the details of its physical implementation. The point is not that particular algorithms don't have a certain implementation independence, that they can be be properly implemented in a variety of different ways. It is that as far as what Clark calls the 'cognitive content' is concerned "... both the perceptual and motor capacities of the system in which implementation occurs are crucial." Our work on building real robots that work reliably in real environments confirms this insight. In particular, our use of motor actions, not just sensor signals, as input to the self-organizing controllers we have developed demonstrate his point about both the perceptual and motor capacities of the system being crucial to its learning of effective competences. Of course, the robots we describe here are not able to develop the cognitive capacities Clark discusses in his paper, nevertheless, we believe that, in this respect, there is no difference between what it takes to learn effective locomotion and simple navigation competences and other, so called higher, cognitive competences.

F.6 Final Comments and Future Directions

The experiments described here, and the self-organizing controllers tested in them, are simple as are the robots used and tasks they perform. This is both deliberate and necessary. It is deliberate in the sense that we wanted to keep things simple so that we could more easily see what was going on—something that is never very easy to do when using real robots operating in the real world. It is necessary in the sense that given our present understanding we are not able to build more complex controllers which work, at least not in real robots. Despite this simplicity we believe that the results we present here offer a significant start to our understanding of multiple competence learning.

Future research must include more extensive testing of the controllers and stage learning scheme presented here. This must include tests in more complex environments and the learning of more competences. We plan to carry out such experiments using robots constructed from second generation of Lego vehicle technology[5] that we are currently developing at the VUB AI Lab in Brussels.

Acknowledgements

The work reported here was supported by a grant from the UK Science and Enginering Research Council (GR/F/5852.3). Other facilities and technical support were provided by the Department of Artificial Intelligence, Edinburgh University. John Hallam and Pete Forster were also involved in the funded project and we acknowledge their contribution.

[5]see [Donnett & Smithers 91] for a description of first generation Lego vehicle technology.

Index