Department of Computer Science

PhD Thesis

Perception-Based Learning for Fine Motion Planning in Robot Manipulation

Enric Cervera i Mateu

Castelló, 1997

Supervisor: Angel Pasqual del Pobil i Ferré

Per a Rosa,
sense el seu suport
aquest treball no hauria segut possible.

The work presented in this thesis has been carried out partially in the CEIT (Gipuzkoa), and mainly in the Laboratory of Robotic Intelligence at Jaume-I University of Castelló (Spain). Support for this laboratory is provided in part by the Comisión Interministerial de Ciencia y Tecnología (CICYT), under the projects: TAP92-0391-C02-0 and TAP95-0710; in part by the EU ESPRIT Programme: EP21007; in part by the Fundació Caixa Castelló: B-41-IN, A-36-IN, and P1A94-22; and in part by the Generalitat Valenciana: GV-2214/94.

Perception-Based Learning for Fine Motion Planning in Robot Manipulation

by

Enric Cervera i Mateu

Abstract

Robots must successfully execute tasks in the presence of uncertainty. The main sources of uncertainty are modeling, sensing, and control. Fine motion problems involve a small-scale space and contact between objects. Though modern manipulators are very precise and repetitive, complex tasks may be difficult –or even impossible– to model at the desired degree of exactitude; moreover, in real-world situations, the environment is not known a-priori and visual sensing does not provide enough accuracy.

In order to develop successful strategies, it is necessary to understand what can be perceived, what action can be learnt –associated– according to the perception, and how can the robot optimize its actions with regard to defined criteria.

The thesis describes a robot programming architecture for learning fine motion tasks. Learning is an autonomous process of experience repetition, and the target is to achieve the goal in the minimum number of steps. Uncertainty in the location is assumed, and the robot is guided mainly by the sensory information acquired by a force sensor.

The sensor space is analyzed by an unsupervised process which extracts features related with the probability distribution of the input samples. Such features are used to build a discrete state of the task to which an optimal action is associated, according to the past experience.

The thesis also includes simulations of different sensory-based tasks to illustrate some aspects of the learning processes.

The learning architecture is implemented on a real robot arm with force sensing capabilities. The task is a peg-in-hole insertion with both cylindrical and non-cylindrical workpieces.

Thesis Supervisor — Professor Angel Pasqual del Pobil i Ferré
Associate Professor of Computer Science
and Artificial Intelligence

Acknowledgments

I am deeply indebted to my supervisor, Angel Pasqual del Pobil, for his constant and generous advice, encouragement, and support. Many of the key ideas in this thesis arose in conversations with Angel, and this work would have been impossible without his help.

I would like particularly to thank Edward Marta and Miguel Angel Serna, for their collaboration in the initial stages of this research, and all the people at the CEIT for their friendship.

Thanks to Jari Kangas, Michael Kaiser, Profs. Kohonen and Dillman, and all the people who kindly helped me during my visits to the Laboratory of Computer and Information Science in Helsinki and the Institute for Real-Time Systems and Robotics in Karlsruhe.

Thanks to my colleagues of the Robotic Intelligence Group for helpful discussions, and to many other people in the Departments of Computer Science, Mathematics, and Technology of Jaume-I University for their support in many aspects of the research: from discussing concepts of statistics, manufacturing the pieces for the experiments, to the invaluable help with LaTeX. Thanks also to the administrative personnel of the department of Computer Science for their help in all those little things.

Finally, I wish to thank my parents for their unconditional support over many years. Thanks for believing in education.

The author has been funded by a grant from the Spanish Ministry of Education (from February'94 to September'97).

Contents

Chapter 1

Introduction

The conjunction of perception and action has been recognized as a key concept in both fields of robotics and artificial intelligence (AI). Brady [1985] defined robotic science as *the intelligent connection of perception to action.* And, for Winston [1992], AI is *the study of the computations that make it possible to perceive, reason, and act.*

AI has failed because there are no working systems. The work in this thesis is motivated by the **engineering goal** of AI which is, according again to Winston [1992], to solve real-world problems using artificial intelligence as an armamentarium of ideas about representing knowledge, using knowledge, and assembling systems.

Our first motivation was a real problem (presented in Sect. 4.4), the extraction and insertion of tools of a machining center and a tool pallet by a robot arm. The uncertainty in the environment made a pure positional approach unfeasible. First, the robot was endowed with a force sensor in order to capture signals which identified contacts. However, the processing of these signals was difficult, and its relationship with the required motion was not clear.

Our attention turned to learning techniques, neural networks and machine learning, to provide the robot and its controlling computer with the capabilities for adequately processing the sensor data (*representing knowledge*) and for managing the complex problem of learning which actions were required according to the perception from the sensors (*using knowledge*).

This approach shed new light onto the peg-in-hole problem (in simulations), and the real implementation was actually carried out in a small robot in our laboratory for a similar task, a real peg-in-hole insertion with non-cylindrical shapes (*assembling systems*); it must be noted that very few cases are found in the literature with 3D real robots.

1.1 Statement of the problem

The basic problem is that *robots must successfully execute tasks in the presence of uncertainty.* The main sources of uncertainty are modeling, sensing, and control. Fine motion problems involve a small-scale space and contact between objects. Though modern manipulators are very precise and repetitive, complex tasks may be difficult –or even impossible– to model at the desired degree of exactitude; moreover, in real-world situations, the environment is not known a-priori and visual sensing does not provide enough accuracy.

In order to develop successful strategies, it is necessary to understand what can be perceived, what action can be learnt –associated– according to the perception, and how the robot can optimize its actions with regard to defined criteria.

1.2 Brief outline of the approach

The basic approach models perception as a feature extraction process, and learning as a refinement of action choices directed towards the maximization of some criteria by means of a mathematically defined feedback.

The robot should act in an autonomous way. The (self-)evaluation of its performance during successive executions of the task should guide its actions in the direction of maximizing its programmed criteria of performance (e.g. minimizing the number of actions to achieve the goal).

Initially, the robot relies on a pre-programmed, suboptimal solution, sometimes a random strategy, provided that there is a reasonable probability of solving the task in a reasonable amount of time. The robot learns if, upon successive task executions, such probability is increased (eventually up to 1) and the average completion time is decreased.

1.3 Issues and goals

Assembly is an important subset of fine motion problems, and the peg-in-hole problem is a classical example (Fig. 1.1). The task is to move the peg from its initial position and orientation down into the hole until the bottom of the peg is resting in the bottom of the hole. Assuming a perfect positioning and control, this task can be accomplished simply by commanding a motion of the peg straight down. The motion is terminated once the position sensor indicates that the peg is in the bottom of the hole.

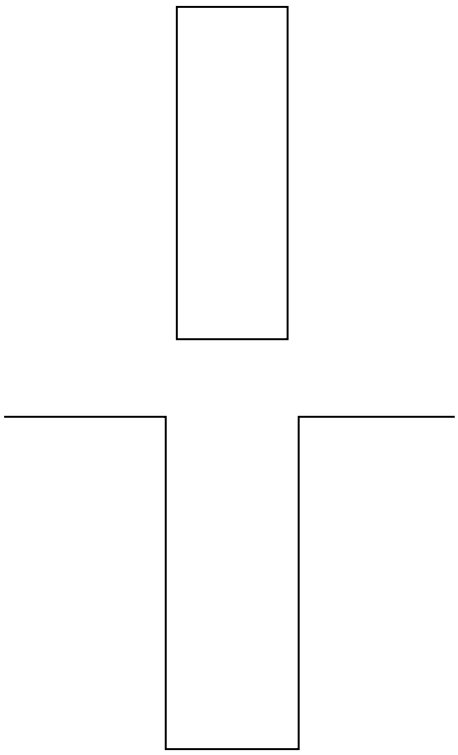


Figure 1.1: A peg-in-hole task

But, if sensing and/or control are not perfect, things get complicated. The precise direction is unpredictable and may vary during motion. In addition, the orientation of the peg is not guaranteed to remain constant. Thus the peg may tilt slightly as it moves. Furthermore, the position sensor is subject to error.

Given these uncertainties, it is clear that the original solution to the peg in hole problem may fail. The extension to three dimensions makes the reasoning process more complicated.

However, people constantly face similar problems every day, and solve them extremely well if compared with robots. Obviously, people have the most sophisticated end-effectors (*hands*) and algorithms (*intelligence*). To be fair, the robot should be compared with persons who have not yet developed their skills, like children.

Children learn basic manipulation abilities in the first months of life. A common educational toy is a set of colorful pieces (prisms, cylinder, cube) and the correspondingly shaped holes. Initially, they found very difficult to put the piece inside. But, slowly, maybe randomly sometimes, pushing, sensing with the fingers, jiggling, trying again, eventually one piece (usually the cylinder is the simplest due to its symmetry) goes in. After numerous repetitions, children are skilled enough to blindly insert every piece in place, they get bored and ask for another toy. But the learnt skills undoubtly remain there to be used in more complex tasks.

1.4 Outline of the thesis

This thesis develops computational tools that provide a robot with skillful manipulation abilities, making possible to learn a fine motion task from an initial, possibly random, strategy plus a basic high-level plan.

In particular, the simulated peg in hole problem is analyzed through the perspective of feature extraction with neural networks, and a related real-world assembly task is monitored using force measurements. Finally a complete robot learning architecture for manipulation tasks is presented.

Chapter 2 reviews the previous work in fine motion planning with uncertainty. Approaches based on force sensing deserve special attention, and real-world experimental results are emphasized.

Chapter 3 provides a fairly detailed view of some learning approaches, including some developed in the course of this thesis. Neural networks are considered for feature extraction, and a reinforcement algorithm is studied for learning actions.

Chapter 4 investigates the peg in hole problem with neural networks.

The extracted features are tested with classification of states. A similar real-world problem is studied with the same techniques, and a plan is build in simulation.

Chapter 5 develops the main contribution of the thesis: a perception-based robot learning architecture for fine motion manipulation. The architecture is actually implemented on a real robot, and experimental results of peg-in-hole problems with different shapes are presented.

Finally, Chap. 6 draws some conclusions, reviews the contributions of this thesis, presents a list of selected publications arisen from this work, and indicates some future lines of research.

Chapter 2

Uncertainty in fine motion

Taking robots out of the shop-floor and into service and public-oriented applications brings up several challenges concerning the implementation of real-time and robust systems. While modern robots are capable of performing a broad range of tasks, the presence of uncertainty in the motion or in the world model makes current hard-coded robot programs fail miserably.

Service robots perform tasks in non-structured environments, where the goals are located with a vision system with limited accuracy. *Adaptability* by means of learning techniques is a requirement for such robots.

Fine motion planning is a domain of robotics restricted to a small scale space and contacts between interacting objects, e.g. sliding along a tabletop, opening a drawer, driving a screw or installing a fuse or light bulb. For gross motion planning uncertainty is not critical. Since a main point is collision avoidance, there is no contact and the clearances between objects can be kept large enough by using adequate spatial representations [del Pobil and Serna, 1995]. Fine motion planning, on the other hand, deals with small clearances and contact. The existence of uncertainty may render a synthesized plan useless.

Assembly is one of the main applications of fine motion. The assembly process is strictly a positioning problem. Perfect knowledge of the parts being assembled and a perfect positioning device would make the task of assembly a trivial matter. Unfortunately, parts are subject to dimensional variations; likewise the realization of a near-perfect positioning device conflicts with cost and flexibility considerations required for a general purpose system. In the service applications described before, position of objects are known very roughly. Departing from the rigid positioning paradigm which dominates robotic manipulation, Mason [1982] pointed out that the only economical way to cope with this problem is to determine the location of the relevant features as the assembly proceeds and adapt the programmed motions to use

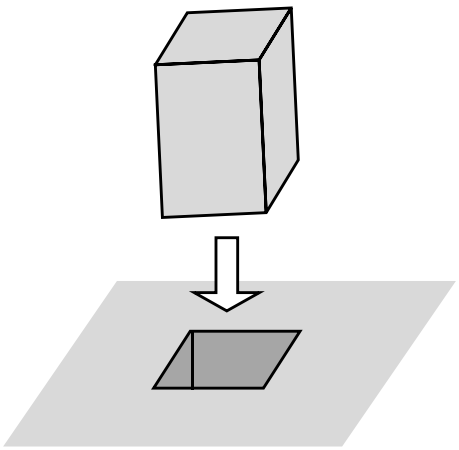


Figure 2.1: Non-cylindrical peg-in-hole task in three dimensions

this information.

A common fine-motion problem which is the main object of study of this thesis is the insertion of a peg into a hole (Fig. 2.1). This task has been analyzed in detail by quite a few authors [Laktionev and Andreev, 1966, Gusev, 1969, Simunovic, 1975, Whitney, 1982], so it has historically become a benchmark for fine motion paradigms, though some different variants are found in the literature: the two-dimensional peg is the simplest problem, though the motivation is its usefulness for the three-dimensional case if cylindrical pegs are considered. The presence of chamfers in the hole, or the assumption of no friction also tend to simplify the problem.

A realistic general fine motion solution should consider pegs of any shape in a real (i.e. uncertain) environment. The work presented in this thesis heavily relies on real experiments. Though the theoretical analysis is restricted to two dimensions, the developed methodology extends to three-dimensional problems, and in the experiments, prismatic pegs with different convex sections (cylindrical, square) are used.

This chapter reviews the related approaches which are found in the literature. First, force control and basic compliant techniques are presented. The second category comprises off-line planning approaches mainly based on geometric considerations but taking uncertainty into account. Next, methods based on contact identification and monitoring are reviewed; and some learning approaches are presented. Finally, a short discussion summarizes their main advantages and drawbacks, and it points out those weaknesses which are intended to be overcome by the methods presented in this thesis.

2.1 Force control and compliance

Robot manipulators are generally controlled by position. Contacts between objects pose the necessity of force sensing to cope with such interactions without damage for the workpieces and the manipulator. Robot force control actually began in the 1950s and 1960s with remote manipulator and artificial arm control. An interesting historical sketch is presented in [Whitney, 1987]. The first computer controls of force feedback date from the late 1960s and 1970s, and various approaches to the creation of strategies emerged, but all the approaches depended on people to formulate the details. In fact, as Whitney [1987] pointed out, no automatic generation of strategies had been achieved at that time.

A commonly used control strategy is the guarded move [Will and Grossman, 1975], which is used to approach and touch an object without producing excessive force after contact is made. This is achieved by moving toward the object slowly while closely monitoring a sensor which can detect contact. Guarded moves can be used for aligning parts with positional uncertainty by setting a small threshold in the direction along which the end-effector of the robot and the other part will first touch. When the end-effector is moved towards the fixed part slowly, the motion will stop when the parts are aligned.

Compliance is another important concept in robot force control. A common characteristic of compliant motions is that the trajectory is modified by contact forces or tactile stimuli occurring during the motion [Mason, 1982]. Compliance occurs either because the control system is programmed to react to force or tactile stimuli, or because of passive compliance inherent in the manipulator linkage or in the servo. Examples of compliant motions are sliding of a piece across a surface, opening of a door, inserting a peg into a hole, grasping an object, and handling eggs. These compliant motion tasks are easily performed by people, but they are very awkward for robot manipulators. Adding compliant behavior to a robot is highly desirable since it would make possible to perform high-accuracy tasks even with low-accuracy position control, for example, the task of following a surface by maintaining a downward force.

Whitney [1982] presented a passive compliance approach, with detailed geometric and force-deformation analyses for rigid part mating, covering the main geometric phases of assembly plus the phenomena of wedging and jamming. During assembly, parts must be supported by fixtures, hands or grippers. These supports have some compliance, either by design or accident. Whitney proposes an unified and fairly general method of modeling supports, and includes the characteristics of such modeled supports in the equations for

mating forces so that the influence of different values of support parameters can be studied. The Remote Center Compliance (RCC) is a passive device which supports parts and aids their assembly. It is capable of high-speed precision insertions. Its major drawback is the lack of flexibility, since the necessary parameters are very sensitive and task-dependent. In addition, the analysis is restricted to two dimensions, and experimental verification is based on the insertion of a cylindrical peg in a chamfered hole.

Hybrid control is a means of implementing active compliant motions. It controls position/orientation along specified degrees of freedom and independently controls force/torque along the remaining degrees of freedom. Mason [1981] formalized this approach by developing a method for synthesizing control strategies for compliant motion, using a precise language to describe force control. The most important aspects of this method are the representations of the task constraints and the manipulator. The use of submanifolds of $\Re^6$ to represent task constraints focuses directly on the most important characteristic of a compliant motion task—the degrees of freedom of the end-effector during the task. Mason points out the relevancy of compliant motions in different domains: mechanical assembly, coordination of multiple manipulators, and stable prehension of an object by a gripper. Inserting a peg in a hole is an example of an important operation in assembly such as parts mating. Since constrained positioning is inevitable in parts mating, compliant motion occurs. Mason's approach decomposes a manipulator trajectory into a sequence of compliant motions joined together by guarded moves.

Asada [1993] demonstrated the limitations of linear compliance and the need for nonlinear compliance. He introduced a multilayer neural network trained with the error backpropagation method [Rumelhart et al., 1986] to represent this nonlinear compliance, and addressed the two-dimensional chamferless but frictionless peg-in-hole insertion task. The major drawback of this approach is that the network is trained in a supervised way, i.e. a training set of input-output (force-velocity) pairs is needed, and it should be provided by an external advisor. This turns to be extremely difficult in real-world three-dimensional problems.

2.2 Geometrical planning approaches

The methods in this section are characterized by the use of geometrical models of the workpieces and the environment. In addition, uncertainties in position and control are explicitly included in the models. The generated plans are normally a sequence of force-controlled motions (guarded moves, compliant motions) described before.

Perhaps the most influential work in this class is due to Lozano-Pérez et al. [1984], who first proposed a formal approach to the synthesis of compliant-motion strategies from geometric descriptions of assembly operations and explicit estimates of errors in sensing and control. The problem of moving rigid objects among other rigid objects is reformulated as the equivalent problem of moving a point among transformed objects in a higher-dimensional space, called the configuration space [Lozano-Pérez, 1983].

Their approach is known as preimage backchaining. Given a motion command, a preimage for a goal for that command is defined as a subset of starting configurations of the robot from which the motion command is guaranteed to reach the goal (goal reachability) and terminate in the goal (goal recognizability). Preimage backchaining consists of iteratively computing preimages of the goal region and preimages of computed preimages taken as intermediate goals, for various selected motion commands, until a preimage contains the initial subset. This general approach, however, raises difficult computational issues which prevent its widespread application. The approach is illustrated with a simple planar example: the rectangular peg-in-hole task.

Erdmann [1986] addresses the problem of computing preimages, introducing the concept of *backprojection*, which is a region in space from which motions in certain directions are guaranteed to enter the goal. Backprojections form the primitive elements from which a planner constructs more complicated preimages.

In another extension to the preimage backchaining approach, Donald [1989] presents a formal framework for computing motion strategies which are guaranteed to succeed in the presence of three kinds of uncertainty (sensing, control and model). The considered motion strategies are sensor-based gross motions, compliant motions, and simple pushing motions. Model uncertainty is represented by position uncertainty in a generalized configuration space. Since it is not always possible to find plans that are guaranteed to succeed, error detection and recovery (EDR) strategies are also investigated. It is shown how this framework is effectively computable for some simple cases, namely the two-dimensional rectangular peg-in-hole task and a gear meshing problem which is also planar. Experimental verification for these problems is described in [Jennings et al., 1989]. The generated plans are executed on a physical force-controlled robot. The tasks are kept planar. The peg insertion plan is quite robust, but the gear meshing plan was less robust since it was subject to a greater variety of unmodeled dynamic and inertial effects.

Most of these approaches are based on geometric models which become complex for non-trivial cases especially in three dimensions [Canny and Reif, 1987]. Natarajan [1988] considers the complexity of motion planning for robots with uncertainty in sensing and control, and demonstrates that com-

pliant motion planning is PSPACE-hard in the presence of such uncertainties. A restricted version of the problem is presented, which is PSPACE-complete. This version concerns motion planning for a Cartesian robot (one that is capable of realizing velocities only along three fixed orthogonal axes) and other restrictions in the scene.

Following Donald's work, Briggs [1989] proposes an $O(n^2 \log(n))$ algorithm, where n is the number of vertices in the environment, for the basic problem of manipulating a point from a specified start region to a specified goal region amidst planar polygonal obstacles where control is subject to uncertainty. The algorithm finds a single commanded motion direction which will guarantee a successful motion in the plane from the start to the goal whenever such a one-step motion is possible. Motions are strictly translational, and no real applications are presented.

Latombe et al. [1991] describe two practical methods for computing preimages for a robot having a two-dimensional Euclidean configuration space. The general principles of the planning methods immediately extend to higher dimensional spaces, but the detailed geometric algorithms do not. Simulated examples of planar tasks are shown.

Laugier [1989] presents a method for automatically generating robust strategies combining sensing operations with small robot movements. The basic idea consists in deducing a fine motion strategy from an analysis of the different ways in which the assembled parts may be theoretically dismantled. It works by reasoning on an explicit representation of the contact space, and it reduces the algorithmic complexity of the problem by separating the computation of potentially reachable positions and valid movements, from the determination of those which are really executable by the robot. Most of the experimentations were executed in simulation, and some of them were implemented using a six DOF robot equipped with a force sensor.

Caine et al. [1989] describe a set of modeling and planning techniques developed to generate robust force control strategies for a certain class of assemblies. Specifically, they present strategies for the chamferless insertion of a planar peg into a hole and the insertion of a three dimensional rectangular peg into a rectangular hole. In the latter case, the set of configurations through which parts must pass is considerably more difficult than for the planar case. Consequently, only a subset of possible configurations is chosen on the basis of a set of heuristics, in order to generate successful assembly strategies.

Xiao and Volz [1989] address the problem of uncertainties by planning robot motions at two levels: Nominal Planning, which assumes no uncertainty, and Dynamic Replanning, to deal with uncertainties that would cause nominal plans to fail. They propose a replanning approach based on knowl-

edge of contacts among assembly parts, and certain constraints on the nominal design parameters, tolerances, and sensor error parameters are enforced. However, the current strategy is limited to translational motions only and it is incapable of handling failures due to orientation errors.

Suárez and Basáñez [1991] propose a methodology to automatically generate an assembly plan based on position and force information, considering several kinds of uncertainty, friction forces and rotational degrees of freedom. Tasks states are defined according to the occurrence of different sets of basic contacts, and state transition operators are established to move through them. The planner is developed for movements on a plane. Extension to more degrees of freedom is theoretically possible, but the authors admit that a hard work is necessary to determine the sensor information (configurations and generalized forces) that can be observed in higher dimensional task states. The proposed planning methodology is applied to a simple task in [Suárez and Basáñez, 1992]. The selected task is the positioning of a block in a corner considering three degrees of freedom, two translations and one rotation.

Geometric planning can be combined with other methods. Though restricted to gross motion planning, Torras [1993] claims that classical geometric planning is more adequate for global planning, whereas neurocomputing is the right approach to deal with a local reactive behavior.

Vougioukas and Gottschlich [1993] describe a methodology for the automated synthesis of compliance mappings. Compliance is considered as a task-dependent mapping from sensed forces to corrective motions which bring the robot closer to its goal. They assume that a nominal path plan has already been developed for the operation and that the path is collision free assuming no uncertainties. They furthermore assume that the plan has been analyzed and that those segments of the path that are prone to collision under the given uncertainty conditions have been identified. For these segments of the path they formulate the compliance mappings that will map erroneous contact forces into incremental corrective motions to be added to the nominal robot velocity (that according to the nominal path). This type of approach is referred to as a two phase approach.

The input requirements for this procedure are the geometric model of the robot and the environment, the model of rigid-body interaction, the initial configuration and nominal velocity of the robot, the uncertainty bounds of the force sensors and of the robot motion, and the goal configuration of the particular task. To identify all possible configurations that may arise during a particular motion due to uncertainty, the procedure essentially simulates the motion of the robot under the given uncertainties, finds all possible contact forces that may arise, and maps them to corrective motions. The simulation

takes place in a sampled configuration space. However, no practical examples are provided, and the complexity of the algorithm is not calculated. This is a tough issue, since, like in other methods based on the configuration space, it might be prohibitive in three-dimensional problems.

A different geometric approach is introduced by McCarragher and Asada [1992] who define a discrete event in assembly as a change in contact state reflecting a change in a geometric constraint. The discrete event modeling is accomplished using Petri nets. Dynamic programming is used for task-level planning to determine the sequence of desired markings (contact state) for discrete event control that minimizes a path length and uncertainty performance measure. In [McCarragher and Asada, 1993a] the method is applied to a dual peg-in-hole insertion task, but the motion is kept planar with only two translations and one rotation being used. The system proves to be highly successful in detecting and recovering from undesirable contact states resulting from misalignment or mismatch between the model and the actual system.

Discrete event models of the assembly tasks are also used in [McCarragher, 1996]. Using such a model, velocity constraints are derived from which desired velocity commands are obtained. The aim is to develop a task-level adaptive controller for use with the discrete event control of robotic assembly tasks. The actual task for implementation is a gear mechanism for a starter motor.

2.3 Contact identification and monitoring

A key problem for most of the planning approaches is the identification of termination conditions for motions. Guarded moves, described in Sect. 2.1, are a simple method for contact detection, but do not provide information about the exact relationship between the parts in contact. Such information is necessary for the precise planning of successive motions. The approaches presented in this section are intended for the identification of contact between parts, mainly based on force information.

Simunovic [1975] analyzed the peg-in-hole insertion process to investigate how much information could be obtained about the state of the insertion process using measurements of the forces produced. The peg is assumed to be cylindrical, so the analysis can be kept in two dimensions. The main results concern the problem of jamming and the necessary conditions to avoid such undesired state.

Considering the fact that part mating strategies consist of sequence of guarded or compliant guarded motions, Desai and Volz [1989] point out the

necessity of verifying the termination of these motions (both successful and unsuccessful) and of identifying the termination state at the end of these guarded motions. They introduce the concept of *contact formations* to describe contacts among parts in a system, aiming at reducing the dimensionality of assembly verification. An elemental contact between two objects is defined as a contact between any two topological elements of the objects, e.g. surface-edge, vertex-surface, edge-edge, etc. A contact formation is then defined as the set of all configurations with identical elemental contacts between the same pair of topological elements. This concept is the base of the contact state that will be used later in this thesis, in the neural approaches to contact identification.

In the same work, they also describe a technique for identifying contact formations in spite of errors in sensing and geometric uncertainties. Since it is impossible to verify termination conditions in general, they introduce the notion of design constraints and they formulate design constraints for which verification can be guaranteed. A two-phase planning approach is used, and the nominal planner is assumed to already exist. The existence of appropriate compliant-motion controllers is also assumed. Verification is done in two phases: a passive phase and, if necessary, an active phase. In passive verification, the contact formation is identified on the basis of the sensed position, forces and moments. If the contact formation cannot be identified unambiguously (due to sensing and geometric uncertainties and approximation of the contact model), active verification is used. More information about the contact formation is collected by making small moves. The algorithms are tested on simple assembly operations such as peg-in-hole.

Hirai and Iwata [1992] present a model-based approach to the recognition of contact states, based on using force information acquired in the mating process. They develop a method for generating the state classifiers based on geometric models of the workpieces. The approach is tested on a real operation of mating the bottom face of a workpiece with the flat top face of the table. The contact states of the operation are represented by enumerating the contacting vertices of the workpiece.

The problem of analyzing ambiguous contact situations in uncertain robotic environments is also dealt with by Spreng [1993]. When any contact is detected, the analysis proceeds by generating a set of hypotheses on the contact, out of which the valid one is detected by testing the feasibility of certain motions, i.e. performing movability tests. A movability test is a small motion which is feasible in case of the presence of one subset of the hypotheses but unfeasible in case of its component. It is assumed that the contact is between the polyhedral end-effector of the manipulator (the moving object) and a polyhedral object of the environment. However, no experimental results are

presented.

Xiao [1993] introduces a method of obtaining sufficient topological contact information based on the current position/orientation sensing data of the objects involved and geometric models, taking into account position/orientation sensing uncertainties. The results can then serve as an initial basis for the application of other sensing means, such as vision and/or force sensing, to confirm whether or not a possible contact situation exists in order to obtain the desired contact information in spite of uncertainties. Contacts between two polyhedra are assumed.

In a later work, Xiao and Zhang [1996] present another approach to obtain the set of all principal contacts that may be formed due to location uncertainties between two objects. The method grows exactly an arbitrary polyhedral object in the three-dimensional Cartesian space by its position and/or orientation uncertainties. The grown region of an object by its location uncertainty could be useful in predicting collisions and thus preventing unintended collisions due to uncertainties during motion planning. In addition, once a collision occurs, it can play a major role in automatically recognizing the topological contact in the presence of location uncertainties of the objects involved. However, exact grown regions may not be easy to compute due to the lack of efficient algorithms and thus approximate representations may be implemented.

A model-based approach to the recognition of the discrete state transitions that occur in assembly processes is presented by McCarragher and Asada [1993b]. The modeling technique incorporates the dynamic nature of assembly. In addition, a qualitative approach is used for signal understanding to highlight the critical components of the force signal. The presented analysis is limited to planar models and planar motions, and the geometric models of both the workpiece and the environment are required. The exact location of the constraint environment is assumed unknown, and both force and moment information are available from appropriate sensors. The peg-in-hole problem is used as the test case.

The qualitative aspects of force sensing are studied by McCarragher [1994]. This work presents an analysis of force data that was generated by a human demonstration of a complex and asymmetrical insertion task. The person was blindfolded and only had force sensors available. The issue is to understand how the force sensory signal is used, i.e., to determine what aspect of the force signal is the basis for a human reaction. Qualitative reasoning is used to understand the force signal by identifying the changes in contact state.

Mimura and Fubahashi [1994] propose a method for identifying unknown contact parameters under the assumption that the shape of the grasped ob-

ject, the contact position and contact conditions are all unknown. It is shown that several active sensing motions are needed to identify contact situations, and then equations to be solved become nonlinear. With their algorithm, the direction of contact can be identified in addition to the position and DOF of the contact.

Suárez et al. [1994] deal with the off-line computation of the sets of reaction force directions that can be expected in the presence of uncertainty for any contact situation during an assembly task. The approach is developed for polyhedral objects moving in a plane, with two translational and one rotational degrees of freedom. It is assumed that the robot has an impedance position/force controller and that the movements are slow enough to make inertias negligible. The method is illustrated through a simple example: positioning a block in a corner.

Nuttin et al. [1995] compare three different learning techniques applied to the estimation of the current contact situation. The experiments are performed over a planar problem with three degrees of freedom: the positioning of a block into a corner. The three learning techniques are: backpropagation neural nets, radial basis function neural nets, and classification trees. Several criteria are used for comparison, including the classification accuracy, speed of convergence, ease of use, compactness of representation, etc.

Another pattern recognition approach to identifying contact formations from force sensor signals is presented by Skubic and Volz [1996]. The approach is sensor-based and does not use geometric models of the workpieces. It works by building a fuzzy classifier, where membership functions are generated automatically from training data. The elements used in the experiments were a square plastic block, a pentagon-shaped block and a 3-pronged power plug.

Dutré et al. [1996] present a general model-based approach to identify geometrical uncertainties and to detect topological transitions in contact situations during compliant motion robot tasks, with emphasis on modeling, on-line uncertainty identification and model validation. The method is based on energy consideration. The work is illustrated by a real-world experiment: a complete assembly of a cylindrical peg into a hole. However, due to the computational complexity, only off-line identification and monitoring are executed.

Hovland and McCarragher [1996] present a method for recognition of discrete events by using dynamic force measurements. Each event is described by a Hidden Markov Model (HMM), which allows for modeling dynamic signals. HMMs represent the contact state transitions instead of the contact states, and the frequency components of the force/torque signals are used as observation symbols of the HMMs. Dynamic motions of the workpiece,

friction and sensor noise are allowed, and there is no requirement for exact knowledge of the positions of the workpiece and the environment. The considered assembly process is the planar peg-in-hole task. Although the method works well in this problem, it suffers from two limitations: the amount of training data required and the amount of on-line data log time.

Discriminant functions and clustering techniques are used in an approach to discretizing sensory data by Sikka and McCarragher [1996]. This is intended for applications such as robotic process monitoring or interpreting human sensory data. The discriminant functions are learned from real sensory data, and hence the approach is adaptive and it also takes into account various task parameters such as friction. The testbed is a contact task which consists of sliding a peg across a surface with holes, but motion is kept planar.

2.4 Approaches based on learning

The methods presented in the previous sections rely on off-line algorithms, based on geometric models, and physical knowledge of the task. Instead of specifying the complete behavior of the robot, learning methods provide a framework for autonomous adaptation and improvement during task execution. Inspired by the way that people (specifically children) learn the manipulation of objects, a suggestive goal is to achieve a robot which, starting from a limited ability, is able to improve its skills from its own experience. Since a pure learning approach would be inefficient, the key is to combine learning strategies with other methods and algorithms to synthesize a robust and flexible architecture for real-world tasks with uncertainty.

Christiansen et al. [1990] describe a robot, possessing limited sensory and effectory capabilities but no initial model of the effects of its actions on the world, that acquires such a model through exploration, practice and observation. Their aim is to explore the potential role of robot learning in extending the flexibility and scope of robot manipulation systems. The experimental task is to manipulate rigid planar objects by tilting a tray containing the objects. Beginning with no prior knowledge of the effects of various tilting actions, the system is able to acquire a discrete model of its actions that leads in some cases to 95% success in moving an object to a randomly selected goal configuration.

An approach to learning a reactive control strategy for peg-in-hole insertion under uncertainty and noise is presented in [Gullapalli et al., 1992]. This approach is based on active generation of compliant behavior using a nonlinear admittance mapping from sensed positions and forces to velocity commands. The controller learns the mapping through repeated attempts

at peg insertion. A two-dimensional version of the peg-in-hole task is implemented on a real robot. The controller consists of a feedforward neural network. The output units are stochastic real-valued units, designed specifically for direct reinforcement learning. The rest of the units are trained with the backpropagation rule. In [Gullapalli et al., 1994] the same architecture is implemented on a real ball-balancing task and a three-dimensional cylindrical peg.

Nuttin et al. [1994] present a procedure for fuzzy controller synthesis with a machine learning tool. The robot task consists of inserting a round peg into a hole. They also present a method for synthesizing a fuzzy controller almost automatically. First, a controller is generated off-line by a classical concept algorithm, and, secondly, it is refined on-line using a reinforcement learning technique.

In order to develop robust, skill-achieving robot programs, Morrow and Khosla [1995] propose the development of a sensorimotor primitive layer which bridges the gap between the robot/sensor system and a class of tasks by providing useful encapsulations of sensing and action. Skills can be constructed from this library of sensor-driven primitives. For the domain of rigid-body assembly, they exploit the motion constraints which define assembly to develop force sensor-driven primitives. Example primitives for assembly are: guarded move, sweep, correlation and accommodation. Experimental results are reported for a D-connector insertion skill implemented using several force-driven primitives.

Kaiser and Dillman [1995] propose a hierarchical approach to learning the efficient application of robot skills in order to solve complex tasks. This approach is not only devoted to skill learning but also includes skill refinement as well as the adaptation of skill activation conditions and the identification of a requirement to generate skills. Since people can carry out motions with no apparent difficulty, the same authors apply their method to the acquisition of sensor-based robot skills from human demonstration [Kaiser and Dillman, 1996], and investigate two distinct manipulation skills: peg insertion and opening a door.

Hovland et al. [1996] present an approach for representing and acquiring human assembly skills for applications in robotic assembly tasks. An assembly task is modeled as a discrete event system in terms of contact configurations involving the object and the environment. The discrete event controller for a skill is represented as a Hidden Markov Model (HMM). The skill is acquired from human demonstration of the task, and so the observation symbols of the HMM are taken from the variables recorded while the task is performed by a human operator. Experiments are performed on a real robot, but only a planar assembly task is considered, thus only three of the

manipulator joints are used. The method is limited like any other method based on a training set. The system may be unable to handle situations not encountered in the training set.

Part mating with a multifingered gripper has not been investigated too much so far. Paetsch and von Wichert [1993] investigate how to find and to realize a complex and useful hand behavior, which represents also a useful strategy to solve the task under the influence of high uncertainty. Their solution consists of several parallelly applied strategies derived from the human (hand) behavior. The employed strategies are: upright keeping, contact force reduction, pushing and shaking. Experiments are reported for real insertion tasks with rectangular and cylindrical pegs, achieving more than 90% of successful insertions. However, the authors point out the difficulty of finding suitable strategies in a systematic and analytical way for other tasks. In addition, the parameters of the single strategies are chosen in a purely heuristic way.

2.5 Discussion

The reviewed collection of fine motion planning methods is a proof of the importance of the problem and the interest raised among the robotic research community. Current methods are far from being satisfactory though: most of the study is restricted to planar problems; they are not always able to produce a plan particularly if the part geometries involved are complex and the uncertainties are large. A heavy dilemma exists between too simplistic models or more reasonable ones which, on the other hand, are too computationally expensive.

As stated by Gottschlich et al. [1994], *challenges still facing this field are how to make fine motion planners that are more powerful, realistic, and efficient than those currently in operation.*

Force control, guarded moves and compliant motion are powerful techniques for low-level controllers, suitable for fine motion tasks involving contacts between parts. Most of the planning approaches use these motions as primitives, combining them in sequences for complex task-solving.

Geometric planning approaches attempt to solve higher-level tasks, but they lack flexibility and are currently limited to planar problems due to their computational complexity for managing uncertainty in 3D environments.

Contact identification methods provide information about the relationship between the parts, aiming at guiding the motion planner. Workpieces with complex geometry, and the uncertainty in location or models, cause sensing ambiguities which require additional strategies for successful identi-

Table 2.1: Classification of some reviewed approaches with regard to the type of implementation and the application domain

Task	Real world	Citation
2D peg-in-hole	No	[Lozano-Pérez et al., 1984]
	Yes	[Jennings et al., 1989]
	Yes	[Desai and Volz, 1989]
	No	[Asada, 1993]
	Yes	[Hovland and McCarragher, 1996]
	Yes	[Sikka and McCarragher, 1996]
	Yes	[Hovland et al., 1996]
2D block in corner	No	[Suárez and Basáñez, 1992]
	No	[Nuttin et al., 1995]
2D motion in conf. space	No	[Briggs, 1989]
	No	[Latombe et al., 1991]
Workpiece on table	Yes	[Hirai and Iwata, 1992]
Opening door	Yes	[Kaiser and Dillman, 1996]
Inserting D-connector	Yes	[Morrow and Khosla, 1995]
3D assembly	No	[Laugier, 1989]
3D round peg	Yes	[Whitney, 1982]
	Yes	[Nuttin et al., 1994]
	Yes	[Gullapalli et al., 1994]
	Yes	[Dutré et al., 1996]
	Yes	[Kaiser and Dillman, 1996]
3D rectangular peg	Yes	[Caine et al., 1989]

fication.

To briefly summarize, a small comparison referencing the experimental achievements of each method is illuminating (see table 2.1). Though many of the reviewed fine motion approaches are implemented in real-world environments, they are frequently limited to planar motions. Furthermore, cylinders are the most utilized workpieces in three-dimensional problems, and the amount of uncertainty is not fully specified.

Throughout the rest of this thesis, new approaches to fine motion problems with uncertainty will be presented. Elements of the reviewed methods are used, together with learning algorithms which are presented in the next chapter. With stress on experimental results, the goal is to demonstrate the performance of the new methods in a real-world three-dimensional insertion task with uncertainty of both cylindrical and non-cylindrical pieces.

Chapter 3

Learning approaches

This chapter reviews the methods which will be used as components of our architecture for perception-based fine motion planning, and it also presents new contributions to the theory of neural networks.

Two neural network models will be described in some detail: Kohonen's self-organizing map (SOM) and Elman recurrent networks. SOMs will be used for unsupervised feature extraction, and Elman networks are suitable for predicting sequences and thus inferring a finite state automaton which represents the temporal dynamics of a process. All of these neural models have been extensively used in the literature [Mira and Sandoval, 1995, Mira et al., 1997].

A contribution of this thesis to the theory of SOMs is presented in Sect. 3.2, in which a supervised training process is used in conjunction with the unsupervised approximation of the probability density functions of each class by a SOM. The resulting system shows a good classification performance in a wide variety of benchmark problems.

In addition to neural networks, this section focuses on the reinforcement learning paradigm, which will be an important component of our learning architecture. One particular algorithm, Q-learning, is thoroughly described. It is a model-free algorithm which learns a policy (rule to select actions from states) by means of autonomous experiences.

The combination of different sound methods is intended to be the base of a simple, robust, reliable and efficient architecture for real agents (robots) in solving moderately complex real-world manipulation tasks.

3.1 The self-organizing map (SOM)

The Self-Organizing Map (SOM) algorithm, introduced by Kohonen [1982], is an unsupervised training process with the objective of identifying a small set of statistically important features which contain the essential information for the task. The importance of features is derived from the statistical distribution of the input signals. In particular, clusters of frequently occurring input stimuli will become represented by a larger area in the map than clusters of more seldom occurring stimuli. The resulting correspondence between signal features and response locations in the layer has many properties of a non-linear, compressed image or *map* of the original signal space and resembles very much the topographic feature maps found in many brain areas.

As opposed to supervised neural networks (like backpropagation), during unsupervised training only the input data is available. Despite this limitation, SOMs are capable of generating ordered mappings of the input data onto some low- dimensional topological structure, which is very useful in the analysis, visualization and abstraction of high-dimensional data. Another capability is the partitioning of input data into subsets (or clusters) in such a way that data items inside one subset are similar while items from different subsets are dissimilar. However, SOMs are not suited for other applications: Kohonen [1995] emphasizes that it has not been meant for statistical pattern recognition, decision or classification processes.

Throughout this thesis, SOMs are used for different purposes. The ultimate objective is the extraction of features from the signal space of a manipulation task (Sect. 5.2.3). The use of SOMs in classification examples in Chap. 4 is intended only for illustrative purposes of its ability to extract such important features: a good, though not necessarily optimal, classification accuracy is an indicator of the suitability of those features for a task which, however, is just a component of the global problem of autonomous fine motion in manipulation.

3.1.1 Kohonen's SOM algorithm

The units (neurons) of a SOM are arranged in an array (generally one- or two-dimensional), which defines the neighborhood relationship between them. This is usually achieved by assigning a location vector r_i to each unit i of the map. The input pattern is described by an input vector x from some pattern space V. The units are fully connected via w_{ij} (weights) to the inputs (Fig. 3.1). A competitive learning rule is used, choosing the winner c as the output unit with closest weight vector, with regard to a defined distance function (usually Euclidean), to the current input x:

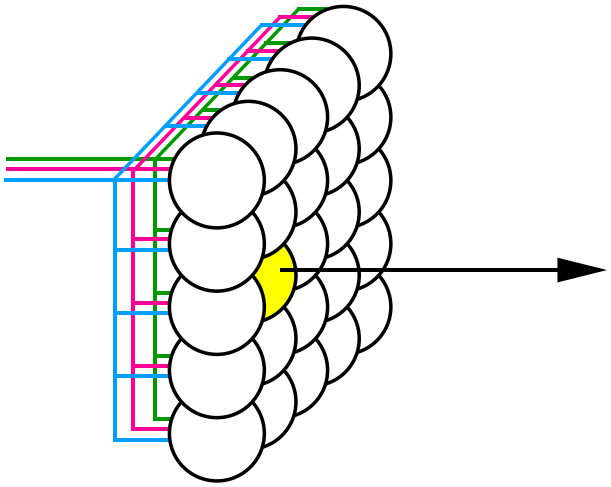


Figure 3.1: Schematic representation of the array of units, fully connected to inputs, and a winner cell

$$|w_c - x| \leq |w_i - x|; \qquad \forall i. \tag{3.1}$$

The learning rule is:

$$\Delta w_i = \eta h_{ci}(x - w_i); \qquad \forall i \tag{3.2}$$

where η is a learning-rate factor, and h_{ci} is the neighborhood function defined over the lattice units. Two simple choices are found in the literature. The first one is just a threshold defined over the distance between units:

$$h_{ci} = \begin{cases} 1 & : \quad |r_c - r_i| \leq r_t \\ 0 & : \quad |r_c - r_i| > r_t \end{cases} \tag{3.3}$$

The second neighborhood function is a Gaussian

$$h_{ci} = \exp\left(-\frac{|r_c - r_i|^2}{2\sigma^2}\right) \tag{3.4}$$

whose variance σ^2 controls the radius of influence. The parameters h, r_t and σ need to be specified appropriately. It turns out to be useful to change these dynamically during learning. Initially, the values of h, r_t and σ are large, and they are gradually decreased as learning proceeds. This allows the network to organize its elastic net rapidly, and then refine it slowly with respect to the input patterns.

The update rule drags the weight vector w_c belonging to the winner towards x. But it also drags the weight vector w_i of the closest units along

with it. Consequently, units that are close neighbors in the map will tend to specialize on similar patterns. We can think of a sort of elastic net in input space that tries to come as close as possible to the inputs; the net has the topology of the output array (i.e., a line or a plane) and the points of the net have the weights as coordinates [Hertz et al., 1991].

Map labeling

This training algorithm is unsupervised, since only input values are used. Frequently, input samples are associated with abstract information (class, state) which is represented by symbolic labels. Each unit is labeled by the test pattern that excites this neuron maximally:

1. Define a counter for each pair (unit, label) and initialize to 0.
2. For all input patterns and their labels:
   (a) Calculate the winner unit for the input pattern, using (3.1).
   (b) Increment by one the counter of the pair corresponding to the winner unit and the current label.
3. For each unit of the map, choose the label with the highest counter as its associated label.

A unit might remain unlabeled if it does not win for any input pattern. The labeling produces a partitioning of the map into a number of coherent regions (clusters), each of which contains units that are specialized on the same pattern. In the next chapter of this thesis, this method will be used for classification of contact states. It is not intended to be an optimal method, since the boundaries between classes are not obtained with a supervised training, but they reflect the probabilistic distribution of the input samples. Classification accuracy obtained by this method may sometimes be rather poor, though this depends on the application (application-specific methods are common in the literature [Hernández-Pajares, 1994]). It should be remarked again that the main purpose of the SOM is monitoring, visualization of high-dimensional data, and feature extraction.

The SOM can be viewed as a nonlinear extension of principal component analysis (PCA). The map selects a subset of independent features that capture as much of the variation of the stimulus distribution as possible, but the difference with linear PCA is that the selection of these features can vary smoothly across the feature map and can be optimized locally. The SOM is also related to the method of vector quantization (VQ). The aim is to achieve

data compression by mapping the members of a large (possibly continuous) set of different input signals to a much smaller, discrete set of code labels.

3.1.2 Applications to robotics

The SOM has been successfully used in other robotic domains different from fine motion planning, which deserve some mention. In [Ritter et al., 1989], an extension of the SOM algorithm is applied to the learning of visuo-motor coordination of a simulated robot arm. A real implementation of the method is described in [Walter and Schulten, 1993]. The method is extended and improved by Ruiz de Angulo and Torras [1997] for the self-calibration of a space robot.

In mobile robot navigation, Kröse and Eecen [1994] present a sensor-based scheme which makes use of a global representation of the environment by means of a SOM. This discrete map is not represented in the world domain or in the configuration space of the vehicle, but in the sensor domain, and it is built by exploration. Simulation results are provided.

DeMers and Kreutz-Delgado [1996] present a method for solving the non-linear inverse kinematics problem for a redundant manipulator by learning a natural parameterization of the inverse solution manifolds with self-organizing maps. The method is demonstrated for the case of a redundant planar manipulator.

A methodology for generating all possible arm postures associated with a specific end-effector position is presented in [Campos, 1996]. The algorithm, as an adaptive self-organizing mapping, learns redundant postures based only on the arm projections from a set of planes placed arbitrarily around the arm.

3.2 Hybrid learning with multiple SOMs

When considering the application of SOMs to the robotic problems of Chaps. 4 and 5, a method for increasing its identification capabilities was investigated. Though finally it has not been used in the implementations, it is a collateral result of this thesis since it is a novel approach to classification tasks with SOMS, and it exhibits a good performance in several tests and benchmarks of quite different domains [Cervera and del Pobil, 1995c, 1997c].

The presented method combines a supervised phase and the usual unsupervised learning process. It is assumed that the problem consists of a set of input vectors, and there is a finite set of output labels, associated with the appropriate inputs (i.e. a classical pattern recognition problem). The output (label) information of the training data is now used during the learning

process, thus performing a supervised learning phase.

Other approaches have been proposed in order to perform supervised learning with SOMs. Kohonen [1995] describes the Supervised SOM, which is a quite different approach. In order to make the SOM supervised, the training vectors are composed of two parts: the first one is the actual input signal; the second part is a vector which encodes the output class. This second part is not considered during recognition. Supervised learning here means that whereas the classification of each input signal in the training set is known, the corresponding class value must be used during training. During recognition of an unknown sample, only its signal part is compared with the corresponding part of the weight vectors. The unsupervised SOM constructs a topology-preserving representation of the statistical distribution of all input data and tunes this representation to better discriminate between pattern classes.

3.2.1 Probability density approximation with SOMs

The key of the proposed approach is that a SOM approximates the probability density of the training set. If the probability densities of each class (denoted by $p(x|C_i)$ for class i) were known, Bayesian decision theory is used to minimize the average rate of misclassifications, assuming that the a priori probabilities of each class (denoted by $P(C_i)$) are also known [Devijver and Kittler, 1982]. Then a sample x would be assigned to class C_i iff:

$$p(X|C_i)P(C_i) > p(X|C_j)P(C_j); \qquad \forall j \neq i. \tag{3.5}$$

Although the distribution of the SOM units is not exactly the same as the distribution of the underlying training set density (for the one-dimensional case the density of output units is proportional to $P(x)^{\frac{2}{3}}$ around point x [Hertz et al., 1991], this does not matter in Bayesian classification where only ratios of class densities are important. Furthermore, the dimensionality of the SOM is less than that of the training set, thus a SOM is a nonlinear projection of the probability density function of the high-dimensional input data onto the usually one- or two- dimensional array of units. For the sake of simplicity, the distance of the input vector x to the best matching unit is considered the measure of the probability density function at that point. A sample x is thus assigned to class C_i if and only if it is closer to a unit of the class map M_i than to any other unit of any other map M_j:

$$\min_{u \in M_i}(|x-u|) < \min_{v \in M_j}(|x-v|); \qquad \forall j \neq i \tag{3.6}$$

Therefore, the probability density function is considered inversely proportional to this distance.

3.2.2 Supervised learning procedure

The proposed, simple, yet powerful idea is based on training a different SOM for each class. Thus, output data from the training set is used to divide all data in several subsets, each one containing only data from one class. This is the supervised part of the procedure. After this, the SOMs will be trained as usual, in an unsupervised way, as described in Sect. 3.1.1. A similar idea was used by Kurimo [1994], where the system is initialized with small SOMs, one for each phoneme class, which are combined and applied to other methods. Another similar system was explained in [Naylor et al., 1988] where one SOM is trained for each speaker. This system is able to classify a speaker but it needs a whole sequence of input data. In our approach, however, classification is performed with a single input pattern. In the experiment reported by Naylor et al. [1988], a whole sequence of inputs were tested on each SOM and the SOM which gave the smallest quantization error was the winner. Our approach is a generalization of this scheme and is able to classify properly not only sequences but also single data samples.

The complete learning process consists of three phases:

1. Division of the training set into as many subsets as different classes.

2. Training of different SOMs, each with only one of the subsets of data, following the unsupervised learning procedure defined in Sect. 3.1.1.

3. Labeling of units of all the SOMs. This can be done in two ways: either labeling each SOM as a whole, thus all its units will be labeled as belonging to the class it was trained with; or all SOMs are labeled jointly with all training data with the labeling method of Sect. 3.1.1. In the experiments, the latter method has been found to be more accurate when there are overlaps between classes.

Now, classification of unknown data is done with all the SOMs together. When an example is presented, the best matching unit of all maps will provide the corresponding class. To evaluate the performance of this hybrid approach, it has been tested with several benchmarks which have been reported in the literature of neural networks and pattern classification. The first one is a synthetic problem; results on other two real problems are presented afterwards.

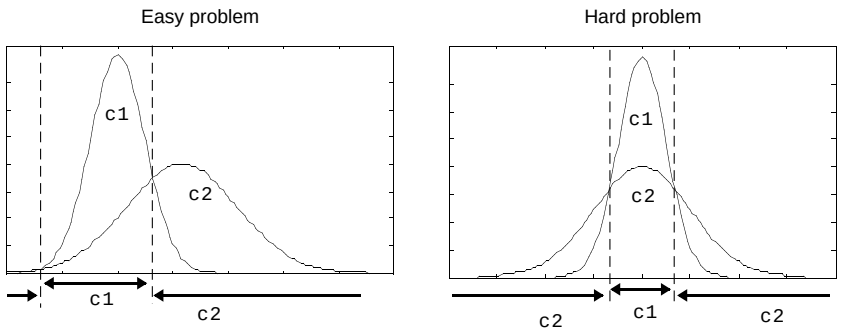


Figure 3.2: Two normally-distributed classes, and optimal class boundaries

Table 3.1: Test results of the two class problem (% of errors)

Easy problem

D	Theory	One SOM	MSOM(1)	MSOM(2)	BP	LVQ
2	16.4	17.9	21.7	18.4	16.4	17.0
4	11.6	18.0	19.3	16.3	12.5	13.1
6	8.4	18.3	16.4	15.9	10.8	10.7

Hard problem

D	Theory	One SOM	MSOM(1)	MSOM(2)	BP	LVQ
2	26.4	27.4	30.3	26.6	26.3	26.5
4	17.6	26.9	24.3	23.7	19.4	18.8
6	12.4	26.3	20.9	20.8	20.7	15.3

3.2.3 A synthetic classification problem

This is a problem introduced by Kohonen et al. [1988] that consists of two normally distributed classes in a d-dimensional Euclidean space with equal class a-priori probabilities. In the easy test the first class has an $N(0, I_d)$ distribution and the second class an $N(\mu, 4I_d)$ distribution with $\mu = (2.32, 0, \ldots, 0) \in \Re^d$. The hard test has the two classes distributed as $N(0, I_d)$ and $N(0, 4I_d)$, respectively. Figure 3.2 depicts the normal distributions in the one-dimensional case, and the optimal decision boundaries for classification.

The dimensions considered are $d = 2, 4, 6$. The SOM architecture is rectangular. The studied size is 6×6, 8×8, and 10×10, thus the number of units is 36, 64 and 100. Two independent sets for training and testing were used. Each set consisted of 2000 randomly chosen samples from each class.

The data and the trained SOMs for each class in the two-dimensional easy

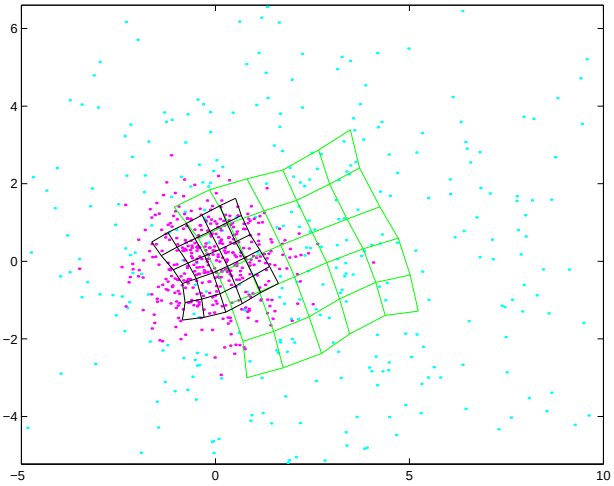


Figure 3.3: Data and SOMs trained in the two-dimensional easy problem

case are shown in Fig. 3.3. A summary of the results is given in Table 3.1. The second column gives the theoretically optimal percentage of classification errors. Results for a single calibrated SOM are shown in the third column. The fourth and fifth columns list the results for our method using Multiple SOMs with separate labeling (MSOM(1)) and joint labeling (MSOM(2)) respectively. The last two columns show the results for backpropagation and LVQ as reported in [Kohonen et al., 1988].

A single calibrated SOM is suited only for the two-dimensional problem, since its performance worsens when a higher number of dimensions is considered in the easy problem. On the other hand, the classification errors of the MSOM approach decrease when adding more dimensions. Classification results are not better than those obtained by purely supervised methods like backpropagation and LVQ, but they keep rather close, especially in the hard problem. It should be noted that only two- dimensional SOMs are used, with which, in principle, it is harder to approximate a normal distribution of higher number of dimensions, due to its homogeneity throughout all the dimensions.

In these experiments, we are using more units than in the other methods: in the backpropagation tests 8 nodes were used in the hidden layer according to [Kohonen et al., 1988] with a number of inputs equal to the dimensionality of the input vectors; the number of output vectors equals the number of classes which is 2 in all tasks. In LVQ, the number of processing units

was chosen to be $5d$ (d is the number of dimensions). On the other hand, algorithm complexity is smaller in the case of MSOM, especially with regard to backpropagation. Moreover, training a SOM in this experiment only takes 25 epochs (presentations of the complete training set). However, training a SOM usually requires fixing the values of some parameters in order to get good convergence and a large number of units might be necessary in order to achieve a good approximation of a tricky class; these are not major drawbacks since the algorithm is computationally cheap. In addition, our approach easily lends itself to parallel implementations.

Theoretically, one should get near-optimal results with a great number of units in each SOM, but SOMs are best suited to data which is distributed in a subspace of the high dimensional original space. Ideally, the SOM number of dimensions should be equal to the dimension of this subspace.

The main advantage of our method is that it simplifies the original problem. Training small networks with subsets of data is easier and faster than training a big network with all the data. Furthermore, each independent SOM might be reused in other problems of classification of its class together with other different classes. Thus, we could get a collection of SOMs and pick the relevant ones depending on the classes involved in a given problem.

Another advantage is that this approach gives a reply to a criticism frequently made about neural networks regarding the lack of significance of the weights of the units, and how the network solves the problem. In our method, each SOM is considered as an approximation of the probability density of a class, the weight links are codebook vectors or representative members of that class. The probability density of the class at point x is inversely proportional to the distance from x to the nearest member of the class. This approximation allows the classification process to be done by comparison of the probability densities of each class, as stated by Bayesian theory.

3.2.4 Experiments on real data

This approach has been tested in other real problems with electronically available data from the Carnegie-Mellon University connectionist benchmark collection.

Classification of sonar signals

This is the data used by Gorman and Sejnowski [1988] in their study of the classification of sonar signals using a neural network. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. The data set contains 208

samples. Half of them were used for training, and the other half for testing. Each sample has 60 components.

We used two SOMs each one size 6×6. The units had 60 inputs. And the SOMs were trained for 500 epochs each. After training and labeling, we got 76.2% and 78.6% right classifications on the test set with separate and joint labeling respectively. When compared to the results given by Ripley [1994] on the same data, our method is almost as efficient as backpropagation or nearest neighbor, and it is similar to or slightly better than other methods like projection pursuit regression, multivariate regression adaptive splines, or classification trees. In the paper by Gorman and Sejnowski [1988], results for different backpropagation networks with variable number of hidden units range from 77.1% to 84.7%. On the other hand, a single calibrated SOM of 12×6 neurons only results in 60.7% classification rate, mainly due to the high input dimensionality.

One should note that the input data has many more dimensions than the SOM. However, due to the dimensionality reduction of the algorithm, classification results are comparable to those of other methods, and they would be considerably improved if more training data were available.

Speaker independent vowel recognition

These data are recorded examples of the eleven steady-state vowels of English spoken by fifteen speakers for a speaker normalization study. After processing, input space to the network was 10-dimensional. Robinson [1989] used these data to investigate several types of neural network algorithms. He used 528 frames from four male and four female speakers to train the network and used the remaining 462 frames from four male and three female speakers for testing the performance.

In our simulations we used 11 SOMs of size 3×3 each, thus using a total of 99 units. Each SOM was trained during 2200 epochs, and the test results of the jointly labeled SOMs are shown in Table 3.2 together with those of other methods as reported by Robinson [1989]. A measurement of the computational resources for each scheme is given in the first column which represents the number of hidden-layer units (in multiple layer networks) or the total number of units (in other cases) used in each simulation. The percentage of correct classifications obtained by our method stands among the best ones in a real hard problem like this one, and far better than the single calibrated SOM, with the same number of units.

Table 3.2: Test results for the vowel recognition problem

Classifier	Hidden/total units	% correct
Single-layer perceptron	-	33
Multi-layer perceptron	88	51
Multi-layer perceptron	22	45
Multi-layer perceptron	11	44
Nearest neighbor	-	56
Modified Kanerva model	528	50
Modified kanerva model	88	43
Radial basis functions	528	53
Radial basis functions	88	48
Single SOM	100	36
Multiple SOM	99	52

3.2.5 Discussion

In this section a method for hybrid learning using multiple self-organizing maps has been introduced. The learning procedure is straightforward, since it only requires the division of the training set in separate subsets, one for each data class. Then, a different SOM is trained in an unsupervised way with one of the subsets. Classification is based on the SOM approximation of the probability density of each class and on the Bayesian decision. Not only do we know what the network computes but also how it does it.

Training is simplified since input data is simpler than the complete set; thus, a complex problem is split into several easier problems. In this approach, the neurons are endowed with significance as approximators of the densities of data. Test results have been presented on both synthetic and real classification problems, which demonstrates that this method is comparable to other pure supervised methods widely used —like backpropagation networks— on hard problems, and it is a great improvement over a single calibrated SOM with the same total number of units.

Two interesting extensions might be possible: the first one would be to use training set data with continuous outputs, i.e. a regression-type problem (now, since the training set class assignments are used, only categorized outputs are addressed). A second extension to take this approach a stage further is imposing a neighborhood relation between the individual SOMs; this might allow, e.g. a consistent visualization of the whole set of maps.

3.3 Recurrent neural networks

A different learning paradigm is briefly reviewed in this section. Its use was motivated by the limitations of learning without taking the past history into account (Sect. 3.5). No theoretical contribution to this field will be presented. Our interest is its use in combination with other learning algorithms which will be presented in the next section.

Feedforward neural networks are universal function approximators: theoretically, any continuous function can be approximated at a desired precision with such a network. However, sequences are frequently found in problems which cannot be dealt with feedforward networks, since their output is always the same for a given input. There are other network architectures that are suited for processing sequences: Minsky [1967] already proved that every Finite State Automaton (FSA) is equivalent and can be simulated by some neural machines assembled by (arbitrarily) interconnecting certain basic elements called McCulloch-Pitts cells [McCulloch and Pitts, 1943].

In practice, the problem faced by an agent is to infer the current state of the world from the current and past observations. The agent ignores the underlying dynamics of the world, and it has to infer a model based on the sequences of observations. An overview of recurrent networks and its relationship with the problem of grammar inference is presented in [Castaño et al., 1995].

Elman [1990] proposed a recurrent neural network architecture which has been proved to have as much computational power as a finite state machine [Kremer, 1995]. The input layer of an Elman network is divided into two parts: the true input units and the context units. The context units simply hold a copy of the activations of the hidden units from the previous time step. The modifiable connections are all feed-forward, and can be trained by conventional backpropagation methods. The recurrent connections from the hidden to the context layer are fixed (Fig. 3.4). Cleeremans et al. [1989] have shown that an Elman network can learn to mimic an existing FSA, with different states of the hidden units representing the internal states of the automaton.

Other simple recurrent models have been proposed by Jordan [1986], Stornetta et al. [1988] and Mozer [1989] which differ in the topology of the recurrent connections; second-order models have been presented by Giles et al. [1992].

There are some different techniques for extracting a FSA from a recurrent network: clustering [Cleeremans et al., 1989], neighborhood techniques [Manolios and Fanelli, 1993] and dynamic state partitioning [Giles et al., 1992]. In this last technique, the range of each hidden neuron is divided into

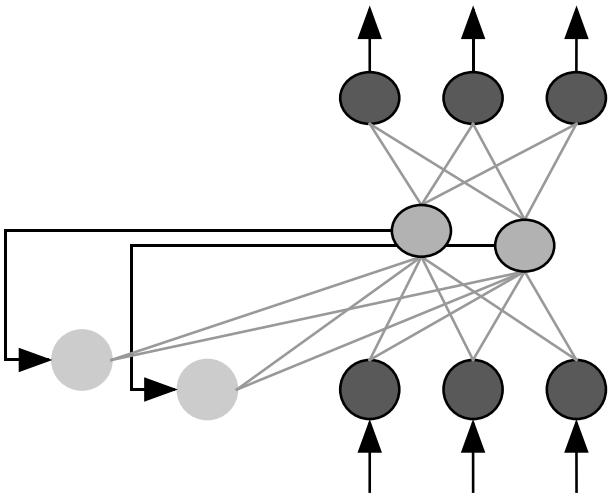


Figure 3.4: Schematic representation of an Elman network with 3 inputs, 2 hidden units and 3 outputs. Connections with the context units are fixed (black arrows). The rest of the connections (gray lines) are trainable

q partitions of equal size. Consequently, a hidden layer of n neurons is partitioned in q^n states. Sequences are presented to the network so that the input symbol is associated to the current partition state and the next partition state they activate. Later, the extracted FSA is reduced to its equivalent minimal state representation using a standard minimization algorithm (see e.g. Sect. 3.4 of [Hopcroft and Ullman, 1979]).

The interest of recurrent networks in our work is the ability to infer an underlying dynamic structure from the sequences of observations. In that case, the agent acquires a state of the world based on current and past information, which can allow it to find a more efficient solution to the problem than if only the current observation is used. In Sect. 3.5, it is shown how recurrent networks can extend reinforcement learning methods to deal with complex partially-observable domains.

3.4 Reinforcement learning

The Self-Organizing Map, described in Sect. 3.1.1, is a method for density approximation and feature extraction; recurrent neural networks are suitable for learning sequences and inferring regular languages. The learning paradigm described in this section departs from the previous ones in its domain of application. *Learning* now applies to an agent (robot) acting in a world (environment), which has to achieve a task (e.g. a fine motion prob-

lem, assembly or part mating). The agent is not provided with the correct solution, but an internal evaluation signal, which indicates if it is doing well or not. Based on this limited information, the learning approaches presented in this section are able to improve the agent's skills and its performance in the task. The interest of the previous paradigms (SOMs and recurrent networks) is its integration with the new learning algorithms to solve particular subproblems (feature extraction, learning past history) which are necessary for the whole learning task to succeed.

3.4.1 Markovian decision problems

In the literature of learning agents, the world (environment) is commonly considered a Markov process with the agent as the controller. In a Markovian decision problem, the agent issues actions, i.e. inputs to a dynamic system, that probabilistically determine successor states. At each time step, the agent observes the systems current state and selects a control action, which is executed by being applied as input to the system. When at state i, the action must be chosen from a given finite set $U(i)$. At state i, the choice of an action u specifies the transition probability to the next state j. The action u depends on the state i and the rule by which we select the control actions is called a policy. Task knowledge might be used to program the action for a given state, but in general the agent should learn to choose the best actions from its own real experience, by means of some mechanism which indicates the quality of the actions performed by the robot (reinforcement).

Each decision results in some immediate cost $g(i, u, j)$ but also affects the context in which future decisions are to be made and therefore affects the cost incurred in future stages. Optimal policies should minimize the total cost over a number of stages. Dynamic programming provides a mathematical normalization of the tradeoff between immediate and future costs [Bertsekas and Tsitsiklis, 1996].

In comparing the available controls u, it is not enough to look at the magnitude of the cost $g(i, u, j)$; we must also take into account the desirability of the next state j. States j are ranked by using the optimal cost (over all remaining stages) starting from state j, which is denoted by $J^*(j)$ and is referred to as the optimal cost-to-go or value of state j. These costs-to-go can be shown to satisfy some form of Bellman's equation [Bellman, 1957]:

$$J^*(i) = \min_u E[g(i, u, j) + J^*(j)|i, u]; \qquad \forall i \tag{3.7}$$

where j is the state subsequent to i, and $E[\cdot|i, u]$ denotes the expected value with respect to j, given i and u. The objective is to calculate numerically the (sub-) optimal cost-to-go function J^*.

Reinforcement learning (RL for short) is a collection of methods used by the artificial intelligence community which are suitable for this class of problems. In the RL community, the concept of reinforcement or reward is more commonly used than the concept of cost introduced in dynamic programming. When the problem is formulated in terms of costs, the objective is to minimize the accumulated costs; however, when using reinforcement or reward, the objective is to maximize the total amount of the received reinforcement signal.

Sutton [1992] has defined RL as *the learning of a mapping from situations to actions so as to maximize a scalar reward or reinforcement signal.* Informally speaking, RL is the problem faced by an agent that learns behavior through trial-and-error interactions with a dynamic environment [Kaelbling et al., 1996]. A more technical definition is given in [Keerthi and Ravindran, 1996]: RL refers to a class of learning tasks and algorithms in which the learning system learns an associative mapping, $\pi : State \rightarrow Action$ by maximizing a scalar evaluation (reinforcement) of its performance from the environment (user).

RL often involves two difficult subproblems. The first is known as the temporal credit assignment problem [Sutton, 1984]. When the learning agent obtains certain outcomes from performing a sequence of actions, it must figure out how to assign credit or blame to each individual situation (or situation-action pair) to adjust its decision making and improve its performance. The second subproblem is the generalization problem (also known as the structural credit assignment problem). When the problem space is too large to explore completely, a learning agent must have the ability to guess about new situations based on experience with similar situations. In the course of learning, both subproblems must be solved.

3.4.2 Q-learning

The basic problem in many tasks is the *delay* of the reinforcement. For example, when the robot is inserting a peg, the reward is attained when the peg is finally inserted though many actions are carried out before, and some of them deserve rewards while others should be penalized. It is a more difficult problem than a pure *associative* reinforcement, when an action is immediately rewarded or punished, so it can effectively be associated to a type of reinforcement.

Q-learning [Watkins and Dayan, 1992] is a RL algorithm that can be used whenever there is no explicit model of the system and the cost structure. This algorithm learns the state-action pairs which maximize a scalar reinforcement signal that will be received over time. In the simplest case, this measure is

the sum of the future reinforcement values, and the objective is to learn an associative mapping that at each time step t selects, as a function of the current state i, an action u that maximizes

$$\sum_{k=0}^{\infty} r(t+k) \tag{3.8}$$

where $r(t+k)$ is the reinforcement signal at step $t+k$. Such an associative mapping is called a policy. In practice, the following expected discount sum is used instead:

$$E\{r(t) + \gamma r(t+1) + \gamma^2 r(t+2) + \cdots\} = E\{\sum_{k=0}^{\infty} \gamma^k r(t+k)\} \tag{3.9}$$

where E is the expectation over all possible behavior patterns of the process. The discount factor γ determines the present value of future reinforcement: a reinforcement value received k time steps in the future is worth γ^k times what it would if it were received now. If $0 \leq \gamma < 1$, this infinite discounted sum is finite as long as the reinforcement values are bounded.

Q-learning was developed from the theory of dynamic programming [Ross, 1983] for delayed reinforcement learning. In Q-learning, policies and the value function are represented by state-action pairs. Formally, for each state i and action $u \in U(i)$ let us define the optimal Q-factor

$$Q^*(i,u) = \sum_j p_{ij}(r(i,u,j) + \gamma J^*(j)) \tag{3.10}$$

where γ is the discount factor and $r(i,u,j)$ is the reinforcement or reward obtained with a transition from i to j under control u. Bellman's equation (3.7) can be written as

$$J^*(i) = \max_{u \in U(i)} Q^*(i,u). \tag{3.11}$$

One should recall that the reinforcement has to be maximized, in opposition to the cost (represented by $g(i,u,j)$ in equation 3.7) which should be minimized. In the following, the notion of reinforcement or reward is always used.

The two previous equations are combined, obtaining

$$Q^*(i,u) = \sum_j p_{ij}(u)(r(i,u,j) + \gamma \max_{v \in U(j)} Q^*(j,v)). \tag{3.12}$$

The Q-learning algorithm works by maintaining an estimate of the Q^* function, and adjusting its values based on actions taken and reward received.

This is done using the difference between the immediate reward received plus the discounted value of the next state and the Q-value of the current state-action pair,

$$Q(i,u) \leftarrow (1-\alpha)Q(i,u) + \alpha(r(i,u,j) + \gamma \max_{v \in U(j)} Q(j,v)). \tag{3.13}$$

The convergence of the process was demonstrated by Watkins and Dayan [1992] under the following conditions. Define $n^k(i,u)$ as the index of the kth time that action u is tried in state i. Given bounded rewards, learning rates $0 \leq \alpha < 1$, and

$$\sum_{k=1}^{\infty} \alpha_{n^k(i,u)} = \infty; \qquad \forall i,u \tag{3.14}$$

$$\sum_{k=1}^{\infty} [\alpha_{n^k(i,u)}]^2 < \infty; \qquad \forall i,u \tag{3.15}$$

then $Q_n(i,u) \rightarrow Q^*(i,u)$ as $n \rightarrow \infty, \forall i,u$ with probability 1. The learning rate schedule of the experiments reported in this thesis has been used by Barto et al. [1995]:

$$\alpha_{n^k(i,u)} = \frac{\alpha_0 \tau}{\tau + n^k(i,u)} \tag{3.16}$$

where α_0 is the initial learning rate. The equation implements a *search-then-converge* schedule for each $\alpha_{n^k(i,u)}$ as suggested by Darken and Moody [1991]. they argue that such schedules can achieve good performance in stochastic optimization tasks. In addition, the schedule satisfies the previous hypotheses of convergence.

3.4.3 Exploration strategies

During the learning process, two opposing objectives have to be combined. On the one hand, the environment must be sufficiently explored in order to find a (sub-) optimal controller. On the other hand, the environment must also be exploited during learning, i.e., experience made during learning must also be considered for action selection, if one is interested in minimizing costs of learning. This trade-off between exploration and exploitation demands efficient exploration capabilities, maximizing the effect of learning while minimizing the costs of exploration [Thrun, 1992].

Exploration schemes can be classified into two families: *undirected* and *directed* exploration. Undirected exploration techniques are characterized by using randomness for exploration. Directed exploration techniques utilize some exploration-specific knowledge for guiding exploration search.

Undirected exploration

The most basic and uninformed undirected exploration technique is called *random* exploration. Actions are selected randomly with uniform probability distribution, and the environment performs a random walk in state space. *Semi-uniform* exploration selects actions with a modified probability distribution: the best action (i.e. the one which maximizes exploitation) is chosen with a fixed probability p_{exp}. With probability $1 - p_{exp}$, a random action is carried out. Consequently, semi-uniform exploration performs exploitation with probability p_{exp} and random exploration otherwise.

Many learning algorithms allow for estimating the exploitation utility of each action separately (e.g. the $Q(i,u)$ values in Q-learning). In *Boltzmann* exploration the utility estimates are used for weighting exploitation and exploration. The probability of selecting an action u in state i is:

$$p(i,u) = \frac{\exp(\frac{Q(i,u)}{T})}{\sum_v \exp(\frac{Q(i,v)}{T})} \tag{3.17}$$

where T is a positive constant value, which controls the degree of randomness and is often referred to as temperature. Its value is gradually decayed from an initial fixed value. When it is close to zero, exploration is turned off and the best action is always selected (see Fig. 3.5). The experimental results in this thesis use a decreasing scheme similar to the presented in [Barto et al., 1995]:

$$\begin{aligned} T(0) &= T_{\max} \\ T(k+1) &= T_{\min} + \beta(T(k) - T_{\min}). \end{aligned} \tag{3.18}$$

Directed exploration

Other exploration methods are presented here for completeness, but they are not actually used in the experiments.

Directed exploration techniques are heuristic in nature, since it is impossible to determine which action will determine the improvement of the resulting controller over time.

Counter-based exploration relies on an adaptive map $c(\cdot)$, counting the occurrences of each state i. This map is used for driving an agent to less explored states, e.g. by selecting the action which maximizes

$$E^{counter}_{explore}(u) = \frac{c(i^t)}{E[c(i^{t+1})|i^t,u]} = \frac{c(i^t)}{c(\hat{i}^{t+1}(i^t,u))} \tag{3.19}$$

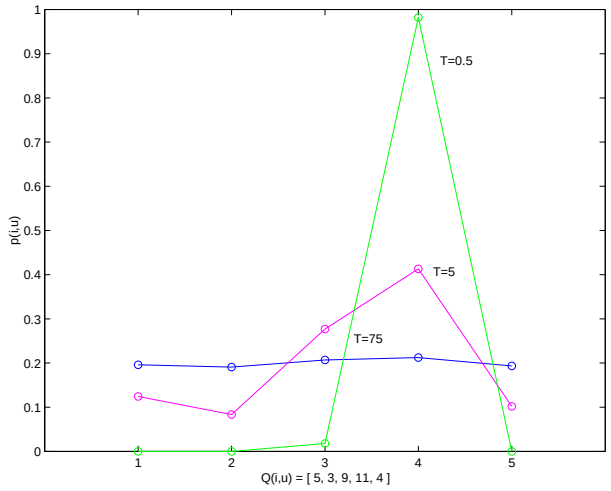


Figure 3.5: Probability of selecting actions using Boltzmann exploration for different temperatures and a fixed vector of Q-values

where i^t denotes the current state, $E[\cdot|\cdot]$ denotes the expected value, and the next state $\hat{i}^{t+1}$ is predicted by the next-state function of a predictive model. This method selects the action which leads to the least visited neighboring state.

The recency of occurrences of states is taken as a basis for exploration in recency-based exploration. This technique favors states for exploration which occurred least recently, e.g. by maximizing

$$E^{recency}_{explore}(u) = \sqrt{E[\rho(i^{t+1})|i^t, u]} = \sqrt{\rho(\hat{i}^{t+1}(i^t, u))}. \tag{3.20}$$

The function ρ measures the recency of a state, i.e. the time during which a state did not occur. This is referred to as exploration bonus in [Sutton, 1991].

Error-based exploration makes the assumption that states or regions in state space with large error are little explored and demand further exploration. Obviously, the efficiency of this rule depends on the heuristic employed for estimating errors.

Exploration and exploitation can be combined in different ways in order to deal with the trade-off efficiently. The most basic and straightforward way is a static linear combination. Actions are selected to optimize a fixed, linear combination thereof, e.g. by maximizing

$$E = (1 - \Gamma)E_{explore} + \Gamma E_{exploit} \tag{3.21}$$

where $\Gamma (0 < \Gamma < 1)$ is a fixed gain, determining the portion of both target functions in action selection. Other more sophisticated dynamic combinations are discussed in [Thrun, 1992].

3.4.4 Reinforcement learning in robotics

Real-world conditions of uncertainty and noise can substantially degrade the performance of traditional control methods. Sources of uncertainty and noise include (1) errors and noise in sensing, (2) errors in execution of motion commands, and (3) uncertainty due to movement of the grasped part with respect to the gripper. The unavoidable problem of uncertainty motivates the introduction of learning strategies in robot control.

The aim of machine learning in robot control is to enable a robot to collect its knowledge on-the-fly, through real-world experimentation. If a robot is placed in a novel, unknown environment, or faced with a novel task for which no a priori solution is known, a robot that learns can collect new experiences, acquire new skills, and eventually perform new tasks all by itself.

In addition to fine-motion applications described in Sect. 2.4 on page 18, reinforcement learning has been successfully used in other robot domains like mobile robotics. Millán and Torras [1992] propose a reinforcement learning architecture that allows an autonomous robot to acquire efficient navigation strategies in a few trials. Their system exhibits important properties: fast learning, it is operational from the beginning, it improves its performance incrementally and it exhibits a high tolerance to noisy sensory data and good generalization capabilities. They report experimental results obtained with a real mobile robot in an indoor environment.

Lin [1993] presents some approaches to overcome the slow learning problem and tackle non-Markovian environments, making reinforcement learning more practical for realistic robot tasks. Integration with artificial neural networks, action models, instructive training instances and hierarchical learning to improve generalization and learning speedup. Memory is used to deal with non-Markovian environments. Results rely on computer simulation, including an agent operating in a dynamic and hostile environment and a mobile robot operating in a noisy and non-Markovian environment.

In [Thrun, 1996] the task of learning from scratch is simplified by considering robots that face whole collections of control learning problems over their entire lifetime. In such a lifelong robot learning scenario, learning tasks are related in that they all play in the same environment, and that they involve the same robot hardware. One particular approach to the lifelong learning problem is the explanation-based neural network learning algorithm (EBNN) [Mitchell and Thrun, 1993]. This is a hybrid learning strategy which con-

sists in learning functions inductively from scratch (just like neural network backpropagation) and learning task-independent domain knowledge, which applies to multiple tasks. Results are presented in [Thrun, 1996] in an indoor mobile robot navigation domain.

3.5 Q-learning and recurrent networks

Throughout the previous sections of this chapter, several learning paradigms have been presented in isolation, together with their main features and application domains. The combination of methods is a possible means of managing more complex learning tasks which otherwise could not be solved. In this section an example of a sensor-based task is presented, together with the methodology for combining two of the learning approaches presented in this chapter: recurrent networks and Q-learning.

The problem is a test case, but it illustrates how the sensing ambiguities might prevent the agent from successfully learning the task. The introduction of recurrent networks allows the agent to take into account the past sensing signals, and learn a good policy for the task.

3.5.1 A sensor-based goal-finding task

This section illustrates an application of Q-learning enhanced with a recurrent neural network to learn a sensory-based exploration task: finding a goal in a two-dimensional grid world. Though very simplified, this task is similar to perception-based fine motion problems. A simple grid is chosen for illustrative purposes, but it could be any physical space, e.g. the contact space. Essentially, the agent does not know its real position but only a sensing measurement. It has to learn a policy from sensing to actions, which allows it to attain the goal with the minimum number of steps.

The agent is not aware of both the real state (location) and the relationship between the sensor measures and the state or action it should perform. This relationship must be discovered in an autonomous way, by performing trials on the task (randomly at the beginning), exploring, and learning the value of each pair (state, action). The indicator of performance is the length of the path at each trial.

Each cell in the grid world produces a sensor reading, but the same measurement may be obtained from different locations. This sensing ambiguity poses a difficult problem, since it invalidates the Markovian hypothesis, and, consequently, degrades the performance of the reinforcement learning methods. In fact, in extremely ambiguous cases (examples will be provided later),

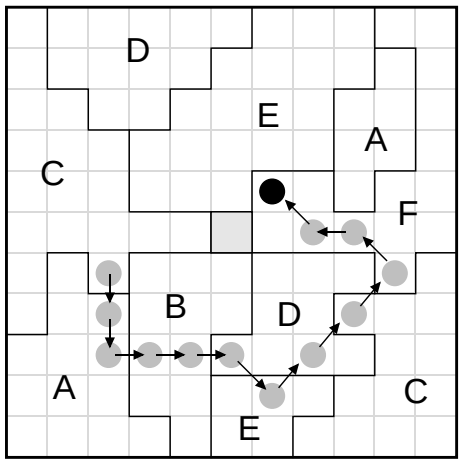
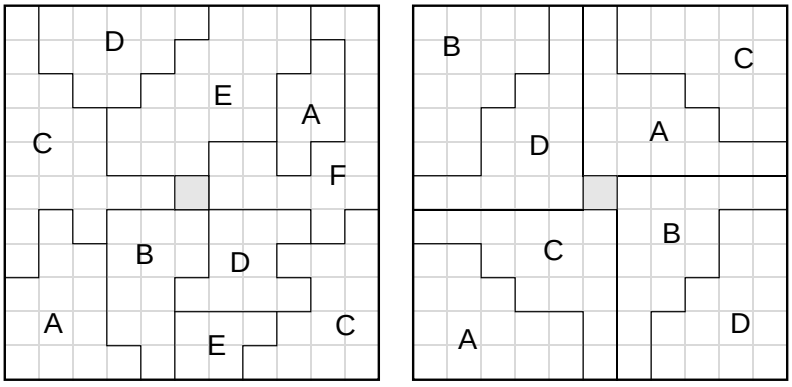


Figure 3.6: Example of agent's motion in a grid world. The goal is the central gray cell. The sequence of sensor readings is: C A A B B D E D C F F F

the learning task turns out to be impossible. This motivates the introduction of recurrent networks: its FSA induction capabilities provide a means of taking the past history of sensor readings into account and overcoming the sensor ambiguities.

The sensor measurement is the only information the agent is aware of: it completely ignores the location of the grid where it stays. Furthermore, at each experiment the agent begins in a different random position: prior to each step, it reads the current sensor measurement and makes a decision about the direction of motion. The agent can move from one cell to each of its 8-neighbors (an example trajectory is depicted in Fig. 3.6). The location of the goal is fixed, as well as the sensor reading associated to each grid location. The world is limited; if the agent tries to move outside, it is just forced to stay in the same cell, though it does not receive any information about this. A trial run ends when the agent gets into the goal, or when a predetermined number of steps have been made.

After each step, the agent's reinforcement algorithm updates its value function based on the previous state, the next one and the reinforcement, which is constant and negative (penalty). The reinforcement signal could really be considered a fixed cost for each step the agent does. The aim is to attain the goal starting from a random location with the minimum number of steps. The agent's performance not only depends on its learning abilities but also on the informative content of the sensor signal with regard to the goal. However, the recurrent network enhances the agent's capabilities: it will be shown how some problems are not solvable without learning the sequences



(a) 1st mapping (b) 2nd mapping

Figure 3.7: Two sensor mappings for the agent's world

of sensor observations.

The presented task is simple at the sensor and actuator levels, but the agent must learn an unknown and possibly complex relationship between states and actions. Starting from a random location, the agent must find its way to the goal (the gray cell) guided only by its sensory information. The sensor measures a single discrete signal with a finite number of different values: two examples of sensor mappings used in the experiments are depicted in Fig. 3.7; in both cases there exist some sensing ambiguities. The second mapping is extremely ambiguous in the sense that, for each reading, there are two different regions which are located in opposite directions with regard to the goal. As a result, it is impossible to learn any optimal action at these regions, since any action that is adequate for one region, is the worst choice in the opposite case.

Let (x, y) be the discrete location (which in fact is the *physical* state) of the agent with respect to a fixed frame of reference. Let (x_f, y_f) be the location of the goal.

Let $\Sigma = \{A, B, C, D, E, ...\}$ be a finite set of sensor measurements. Let $S : (x, y) \rightarrow \Sigma$ be the sensing function, i.e. the mapping from the agent's location to the sensor measurement which is observed at such location (observed state).

Let A be the finite set of actions (discrete motions) that the agent can perform. If the agent's decision is based only on the current observed state, one q-value is stored for each pair (Σ, A), and the action whose value is maximum for the current state is chosen. The problem is that different states are observed in the same manner, i.e. their sensor measurements are

equal. However, the best action to attain the goal from each state can be quite different, and the learning algorithm learns a compromise between those actions, or, if the actions are totally opposite, it cannot learn anything at all (as it will be shown in the examples).

If the current state is ambiguous, the agent needs more information in order to make a sensible choice. This information, in the absence of other sensors, can only be acquired by the analysis of past observations.

3.5.2 A finite state model of the task

During each walk of the agent through the grid, a sequence of sensor symbols is generated, one symbol at each step. This sequence is a string $s \in \Sigma^*$, and the set of all possible generated strings is a language $L \subseteq \Sigma^*$. Since the number of states is finite, the language is regular, and the problem can be modeled by a finite state system. In such systems, the state summarizes the information concerning past inputs that is needed to determine the behavior of the system on subsequent inputs.

Let us define a finite state automaton (FSA) for that language by the 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is the input alphabet, q_0 is the initial state, F is the set of final states, and δ is the transition function mapping $Q \times \Sigma \rightarrow Q$ [Hopcroft and Ullman, 1979]. Let it be named *cell-FSA*:

- For each location (x, y), let $q[x, y] \in Q$ be a state of the automaton.
- Let q_0 be a distinct state of Q, not associated with any location.
- Let the alphabet Σ be the finite set of sensor measurements.
- Let $F = Q - \{q_0\}$, i.e. all the states except the initial one.
- Finally the (non-deterministic) transition function δ is defined as follows: For all $(x, y) \neq (x_f, y_f)$
  - Define $\delta(q_0, \lambda) \rightarrow q[x, y]$
  - $\forall (x_i, y_j) \in neighborhood(x, y)$, define $\delta(q[x, y], S(x, y)) \rightarrow q[x_i, y_j]$
  - If (x, y) is a bounding cell, define $\delta(q[x, y], S(x, y)) \rightarrow q[x, y]$

First, an *empty* transition is defined from the initial state to each other state, since the agent can start from any location and it has not sensed anything yet. Secondly, upon reading the sensor measurement of the current

location, the agent moves to a neighbor cell. Finally, if the agent tries to move outside the bounds, it stands in the same location.

Every state except q_0 is a valid final state, since the agent can stop after a number of steps without reaching the goal. The goal state has no further transitions: the agent stops when it reaches it.

This is a non-deterministic FSA with as many states as cells in the grid world, and it is equivalent to a deterministic FSA (Chap. 2 of [Hopcroft and Ullman, 1979]). The number of states of the system is important, since the Q-learning algorithm needs to store the q-values of each pair (Q, A). The number of states can be reduced if a simpler yet more general automaton is built, by considering all the neighbor cells sharing the same sensor measurement as a single state of the automaton. This notion is valid as long as cells are grouped in regions or clusters and the number of these regions is much smaller than the number of cells (in the examples depicted in Fig. 3.7 there are 120 cells but only 10 and 8 regions, respectively, excluding the goal cell).

Let $R = \{r_1, r_2, \ldots, r_n\}$ be the set of regions or clusters of grid cells such that $r_i = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ and it holds that

$$\forall r \in R, \forall (x, y) \in r, \exists! s \in S / S(x, y) = s \tag{3.22}$$

i.e. all the cells share the same sensor measurement and it also holds that each region is connected. The goal defines a particular region $r_f = \{(x_f, y_f)\}$. Let us define a new FSA over the set of regions:

- For each region $r \in R$, let $q[r] \in Q$ be a state of the automaton.
- Let q_0 be a distinct state of Q, not associated with any region.
- Let the alphabet Σ be the finite set of sensor measurements.
- Let $F = Q - \{q_0\}$, i.e. all the states except the initial one.
- The (non-deterministic) transition function δ is now defined as follows: $\forall r \neq r_f$
  - Define $\delta(q_0, \lambda) \rightarrow q[r]$
  - Define $\delta(q[r], S(r)) \rightarrow q[r]$
  - $\forall r' \in neighborhood(r)$, define $\delta(q[r], S(r)) \rightarrow q[r']$

The behavior of this FSA is defined by the topology of the grid world, i.e. the connections between regions of cells. Let it be named *topology-FSA*. Intuitively, one can see that any sequence recognized by the cell FSA is also recognized by the topology FSA. The opposite may not be true: in the cell

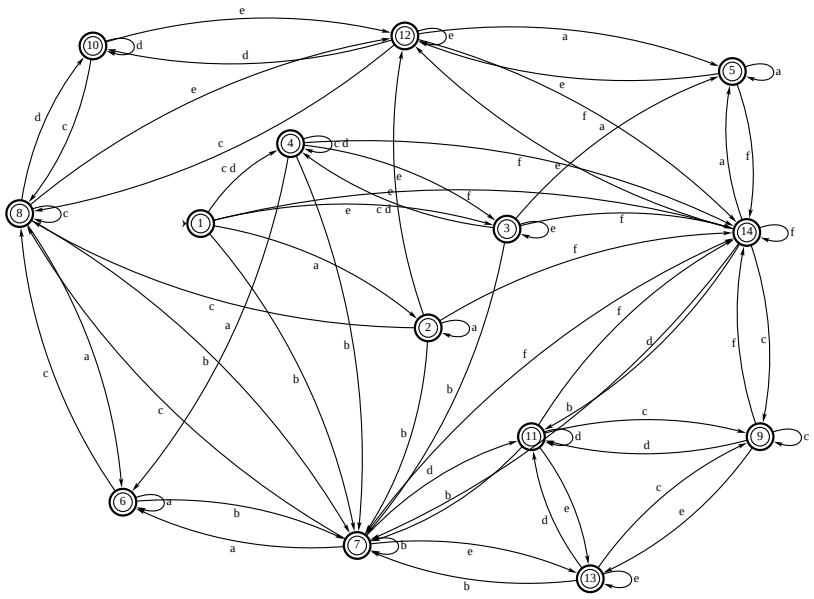


Figure 3.8: Minimized topology FSA defined by the first sensor mapping

FSA, transitions from one region to other are only defined for those cells which bound the region. Thus the language (set of sequences) recognized by the topology FSA is more general than that of the cell FSA, i.e. $L(cell - FSA) \subseteq L(topology - FSA)$.

Deterministic and minimized versions of the topology FSAs for the example mappings are depicted in Figs. 3.8 and 3.9.

The methodologies previously described for building FSAs assume a complete and perfect knowledge of the world, which is clearly an impossible assumption in any real task. More realistically, the agent should build an internal representation of the FSA by means of an inference process, e.g. training a recurrent neural network (introduced in Sect. 3.3) with sample sequences, which can be obtained by random walks across the grid. Once the network learns an accurate prediction of the sequence, the FSA is extracted and, if necessary, determinized and minimized. With this scheme, the number of stored q-values in the learning algorithm depends on the number of regions, not the cells.

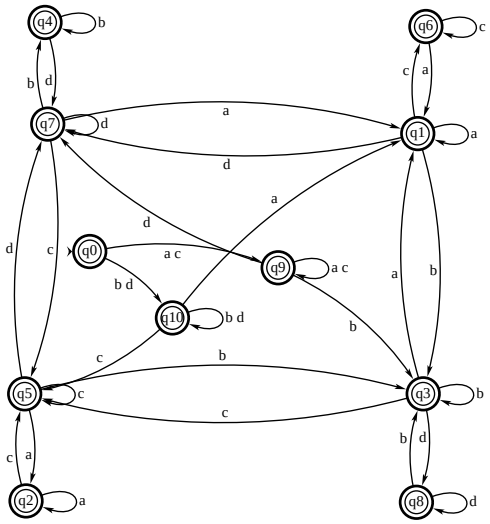


Figure 3.9: Minimized topology FSA defined by the second sensor mapping

3.5.3 Simulation results

The presented approaches are illustrated with computer simulations of the sensor-based goal-finding task. Two possible sensor mappings are used in the simulations (Fig. 3.7). Each location of the grid is associated to a static sensor measurement, but two distinct isolated locations may produce the same sensor value (an extension for stochastic values is possible, since there are recurrent networks capable of learning regular grammars from noisy strings [Carrasco and Forcada, 1995]).

Sensing ambiguities

Sensor measurements are ambiguous and the physical state of the system (in this particular case the position or whatever, e.g. the contact state) cannot be directly obtained from the current sensor signal. Despite this ambiguity, the learning algorithm should be able to find the best action (the one which leads to the goal in the minimum number of steps on average) for those states which are not replicated, in the first mapping, i.e. those regions whose sensor measurement is different from the rest. In the second mapping, configuration of the regions is even more ambiguous, and no optimal action can be learnt

by using only the current symbol information. However, the agent can infer a finite state model of the underlying structure of both mappings by means of randomly walking and building strings of sensor measurements.

Results for the learning task which consider only the current sensor symbol as the state are depicted in Fig. 3.10. Three independent runs of 1500 trials each are shown. The plots represent the number of steps to attain the goal, with a limit of 100 steps if not found, versus the number of trials. A moving average window of 25 consecutive values is used to smooth the data.

In both mappings, Q-learning (Eq. 3.13) with Boltzmann exploration (Eq. 3.17) was used. The learning rate is scheduled as Eq. 3.16, with the initial value $\alpha_0 = 0.5$ and $\tau = 300$. Since the number of steps is limited, no discount is necessary ($\gamma = 1$). The temperature is scheduled as Eq. 3.18 with the values $T_{\min} = 0.5$, $T_{\max} = 75$ and $\beta = 0.992$.

Each trial starts with the agent in a random location and finishes either when the agent reaches the goal or when 100 steps are carried out. In the first mapping, the algorithm converges to a good solution in one of the three trials; the other two show a performance increase affected by an oscillating behavior. The second mapping confirms the intuitive impossibility of learning any sensible action due to the completely ambiguous layout of the regions. The agent cannot improve at all from the initial random strategy.

Learning with FSAs

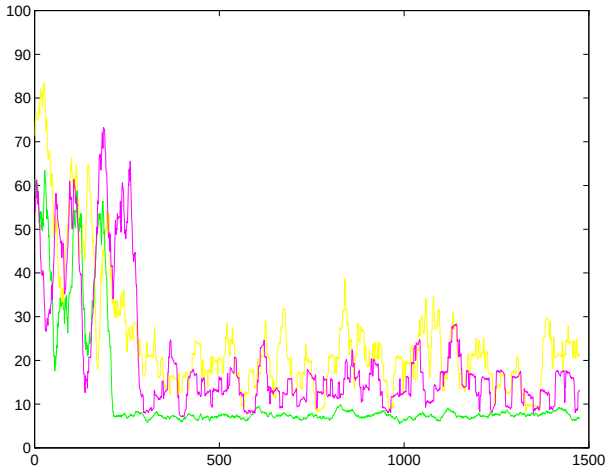
To demonstrate the effectiveness of using FSAs in these learning tasks, the topology FSAs defined in Sect. 3.5.2 are used to determine the state of the agent. The parameters of the learning algorithm and exploration schedule are the same as in the previous experiments.

Experimental results are depicted in Fig. 3.11. The increase of performance is dramatic, specially in the second mapping which previously was impossible to learn. However, the FSAs have been constructed from the knowledge of the world, which usually is not available. Automatic inference of FSA is necessary for a realistic implementation of an autonomous agent.

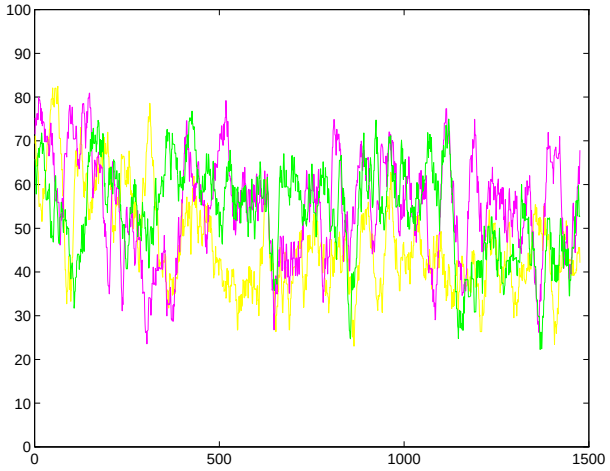
FSA inference with recurrent nets

Previous results have demonstrated the need of taking the sequence of sensor readings into account to determine the state. The learning process is splitted into two phases:

1. Inference of FSA. The agent walks randomly through the grid, recording the sensor sequences and training a recurrent network for predicting the next sensor reading. This network learns an internal representation

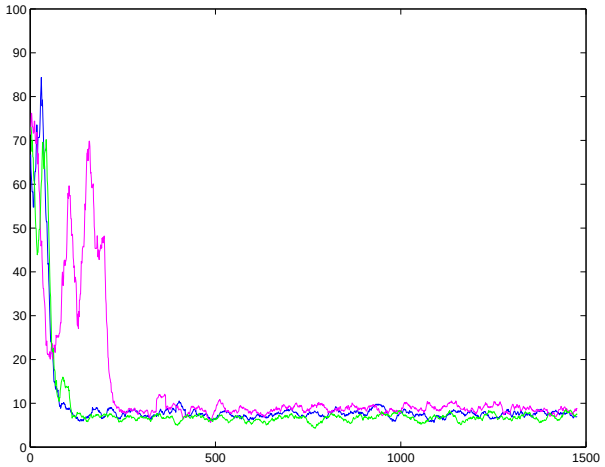


1st mapping

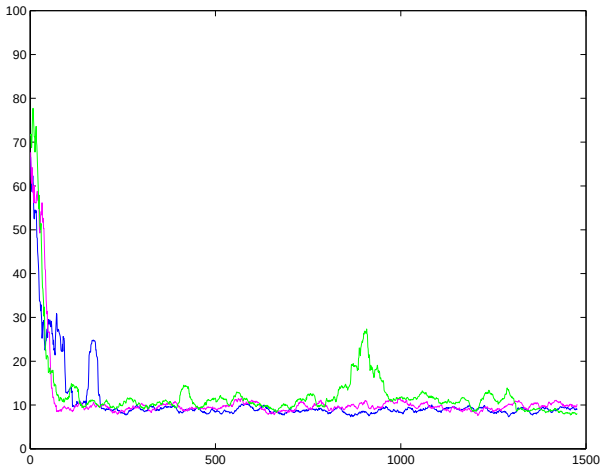


2nd mapping

Figure 3.10: Steps to goal vs. trials when using only current sensor measurement



1st mapping



2nd mapping

Figure 3.11: Steps to goal vs. trials when using topology FSAs

of the FSA which is extracted with any suitable method, determinized and, if necessary, minimized.

2. Action learning. Once the FSA is extracted, the agent begins to learn the actions, with regard to the state determined by the FSA. The learning algorithm is the same that in the previous experiments; only the determination of the current state is different.

In the first mapping, the network has 6 input units (one for each symbol, using *1-of-N* codification scheme), 2 hidden units and 6 outputs. The training set consists of 100 random sequences of variable length (less than 30 symbols), and the network is trained to predict a symbol in its outputs based on the previous one in its inputs. A validation set of other 100 random sequences is used. The network is trained with the backpropagation algorithm with momentum for 150 epochs. The number of epochs was determined by the evolution of the squared error on the validation set. The learning rate is 0.04 and the momentum factor is 0.7. No exhaustive search for optimal parameters was made; we found that the trained network was not very sensitive to small changes in these parameters. The final Mean Square Error (MSE) was 0.54 for the training set and 0.59 for the validation set. Additional hidden units did not improve the network performance very much, and were too complex for extracting the automata.

The FSA was extracted by *dynamic state partitioning* [Giles et al., 1992]. 100 random sequences of variable length (less than 100 symbols each) were presented to the network. The range of each hidden unit $[0, 1]$ was divided into 6 equal intervals. The states of the FSA (up to 36) are the activated intervals and the transitions are defined by the symbols which cause the changes between intervals of activations. The extracted FSA was determinized and minimized to get a final FSA with 23 states and 113 transitions, which is depicted in Fig. 3.12.

Experimental results for the learning task with this FSA are shown in Fig. 3.13. The parameters of Q-learning and the exploration scheme are the same than in the previous experiments. The agent achieves a good convergence in all the runs, though not as fast as that obtained with the topology FSA (Fig. 3.11).

The choice of the number of hidden units and the number of intervals in state partitioning is relevant to the final performance. Unfortunately, there is no automatic determination of these values. Experimental results with different values are depicted in Fig. 3.14. The quality of the learning process depends on the extracted FSA, though a similar behavior is observable in these other cases.

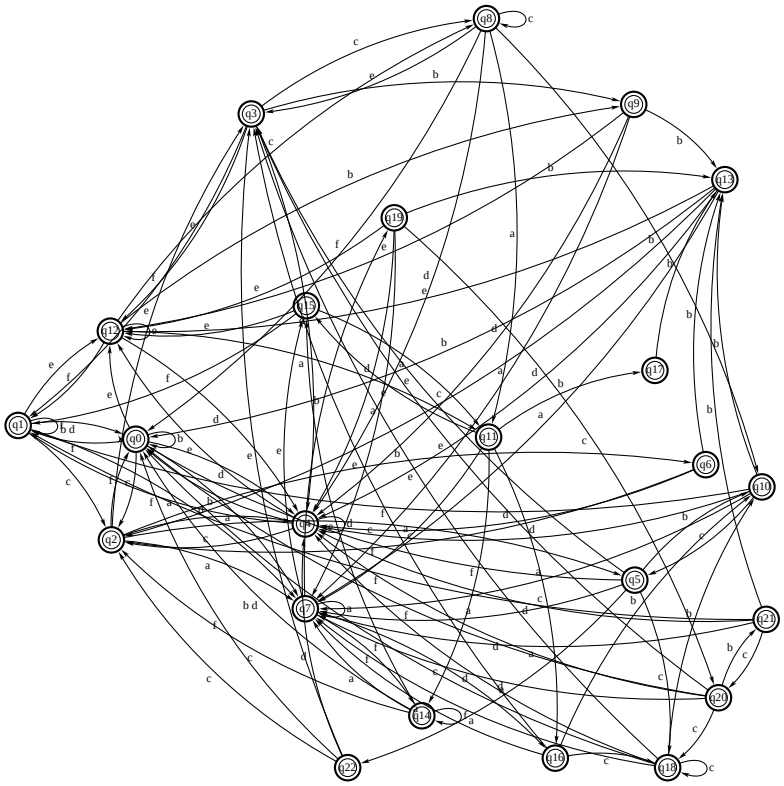


Figure 3.12: Minimized FSA extracted from an Elman network

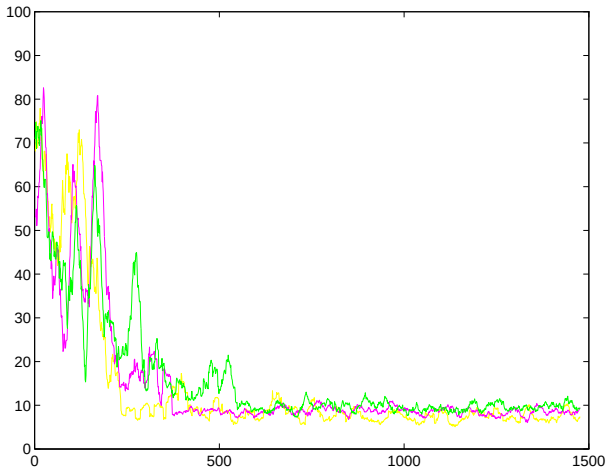
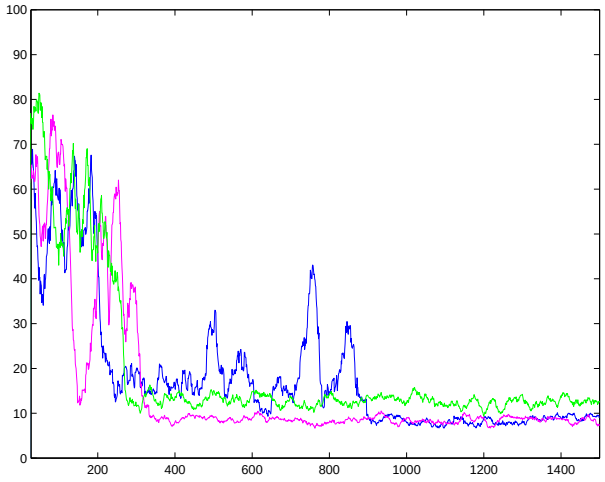


Figure 3.13: Steps to goal vs. trials using the FSA extracted from a 6-2-6 Elman network with 6-interval partitioning for the first mapping

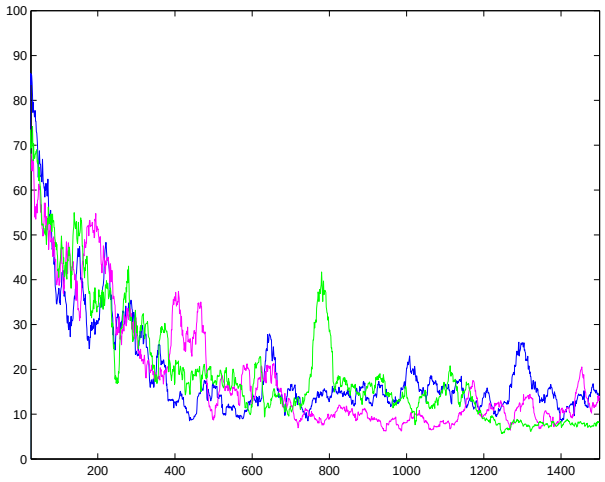
In the second mapping, the Elman network had 4 inputs only (symbols A to D), 2 hidden units and 4 outputs. The training and validation set, and the training parameters were the same as before. The final MSE was 0.46 for the training set and 0.47 for the validation set. The FSA was extracted by partitioning each hidden unit in 5 intervals, and testing the networks with 100 sequences of length less than 30 symbols. The final minimized FSA with 13 states and 48 transitions is depicted in Fig. 3.15.

Experimental results for the learning task of the second mapping with this FSA are shown in Fig. 3.16. The parameters of Q-learning and the exploration scheme are the same than in the previous experiments. The agent achieves a good convergence in all the runs, with a dramatic increase over the poor performance of the system based on current sensor information (Fig. 3.10), and not far from that obtained with the topology FSA (Fig. 3.11).

The choice of the number of hidden units and the number of intervals in state partitioning is also relevant to the final performance. Experimental results for FSAs extracted from the same Elman network, with partitioning in 4 and 6 intervals are shown in Fig. 3.17. With 4 intervals, the learning process slows down and is less stable. On the other hand, with 6 intervals, though there is a small improvement in the converged value, the system takes more trials to achieve this convergence.



6-2-6 network, 5-interval partitioning



6-4-6 network, 3-interval partitioning

Figure 3.14: Steps to goal vs. trials when using other extracted FSAs in the first mapping

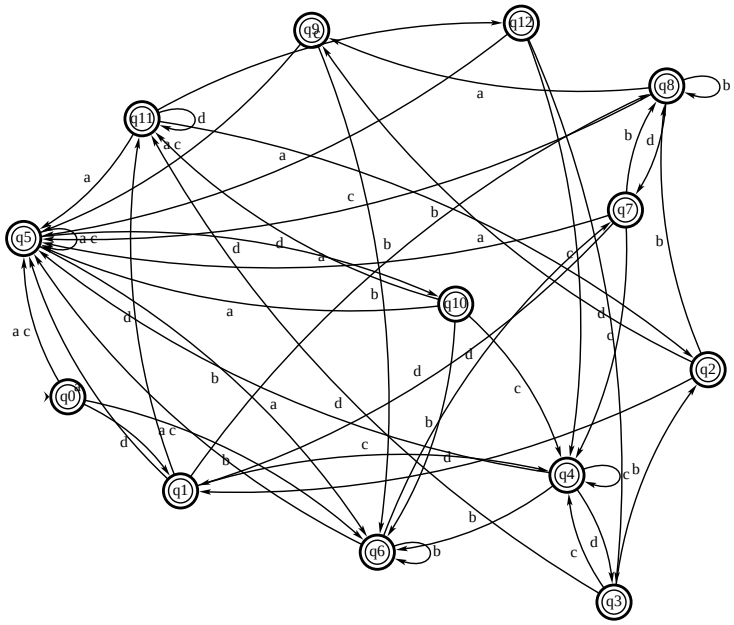


Figure 3.15: Minimized FSA extracted from an Elman network in the second mapping

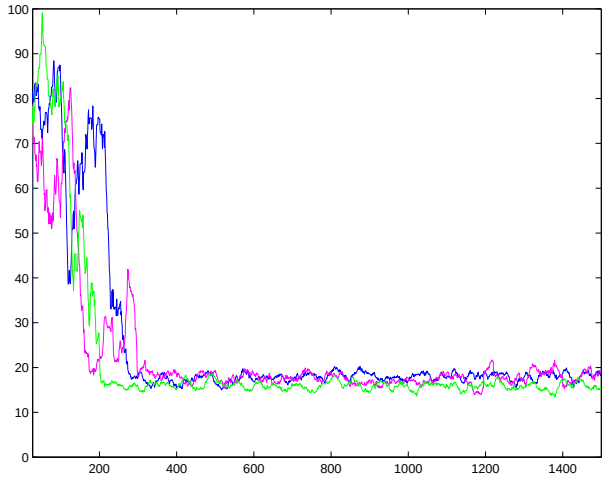
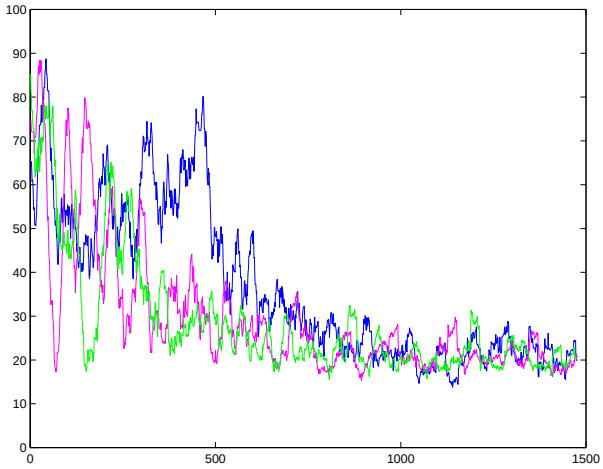
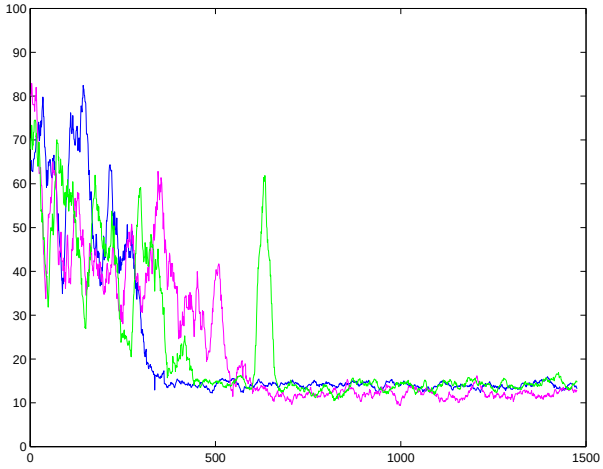


Figure 3.16: Steps to goal vs. trials using the FSA extracted from a 4-2-4 Elman network in the second mapping, with 5-interval partitioning



4-2-4 network, 4-interval partitioning



4-2-4 network, 6-interval partitioning

Figure 3.17: Steps to goal vs. trials when using other extracted FSAs in the second mapping

Discussion

It has been experimentally demonstrated that the inference of FSA can help to improve dramatically the learning capabilities of an agent in a sensory-based goal-finding problem. Even a task which is not solvable if only current measurements are considered, can be successfully learnt with the help of the inferred FSA.

Both the inference and the learning process are automatically performed by the agent. However, a number of parameters have to be manually tuned by an external operator. An interesting extension would be to perform simultaneously both learning tasks: the FSA and the action learning.

The choice of an Elman network is not unique, and more sophisticated (and complex) methods for extracting FSA are available. In particular, automatic methods for the extraction of FSA are needed, which guarantee the best possible performance of the extracted automaton.

Chapter 4

Contact identification with SOMs

This chapter and the next one are devoted to the application of the learning approaches presented in Chap. 3 to fine motion robotic problems. Throughout this chapter, two problems of contact identification and monitoring are dealt with: the insertion of a peg into a hole, which is studied in simulations using only two dimensions with translation and rotation; and a real insertion task in a flexible manufacturing system [Cervera et al., 1995a, 1996]. Self-organizing maps are used in both problems for contact identification, monitoring and error detection.

Sect. 4.5 goes a step beyond the identification problem since it shows how to build a perception-based plan with some recovery capabilities based on states detected by a SOM. In the next chapter, the approach will be pushed a stage further with the integration of SOMs and reinforcement learning to develop an architecture for solving complex fine-motion problems in real-world environments.

4.1 Monitoring with SOMs

The Self-Organizing Map is a useful tool for the analysis, visualization and abstraction of high-dimensional data. A number of applications of SOM to process monitoring, error detection, system assessment and analysis in quite different domains can be found in the literature. We are interested in extending these methodologies to the field of robotic fine motion planning, specially in contact identification and monitoring.

Tryba and Goser [1991] investigate an application of the SOM for process monitoring and process control in chemistry: the SOM learns vectors with

components from on-line measurements of the process, and the goal is to optimize the quality parameters of the output material of the process and the process itself.

Another process-error detection experiment is reported by Alander et al. [1991]. A sequence of SOMs is used to estimate the distribution of the correct processes and error detection is based on the observed deviation of a process from this estimated distribution. Though the initial application is a rather simple process error detection (a coffee making process), their aim is an assembly robot error detection process.

Kasslin et al. [1992] use a SOM to detect operational states of a device. The features used in the map are the measurements from the device describing its operational and environmental parameters. The quantization error can be used to detect a fault, and visual inspection of the map and the trajectory depicts the state changes clearly.

The application of SOMs to power system static security assessment is studied by Niebur and Germond [1992]. Active and reactive line powers are selected as components of the input vectors, and the regions of the map which correspond to a flow constraint violated in a specific line are determined. The secure operating limits for injected powers can be determined with respect to a set of contingencies. The network is further capable of generalizing the trained cases and therefore of classifying operating states not encountered during the training phase.

In a rather different application domain, Martin del Brio and Serrano-Cinca [1993] show the capabilities of SOMs for the analysis and representation of financial data and for aid in financial decision-making. The model is applied to some practical financial cases, making use of real data from the Spanish economy. In their opinion, the conclusions drawn and the methodology used can be easily extended to data processing in other fields.

Representation and identification of fault conditions is explored by Vapola et al. [1994], in an application of an anesthesia system. The existence of any fault condition is detected using a set of so-called absolute features. After that, only if the existence of the fault has been detected by means of absolute features, the fault condition is identified using the differential features.

4.2 Monitoring fine motion tasks

Our aim is to use SOMs in fine motion and contact monitoring. The first decision is which signals are chosen as inputs to the map: position or forces. Changes of contact state in fine motion tasks are caused by small displacements, but changes in forces are significant, since these signals are more

robust against uncertainties. Forces and torques are measured by sensor devices attached to the robot wrist. The output of such force sensors is made of six signals: the three spatial components of the force and torque vectors acting on the wrist. These signals are discretized and sent to the computer for further processing.

The training process (Sect. 3.1.1 on page 24), modifies the weights according to the input stimuli, which consists of samples of force signals registered during task execution or simulation. The units will learn different values of signals and, after training, will become more active when presented with those, or similar, signals. Thus the network will be organized in regions of units, or clusters which contain the main features of the input signal, and which could be related to the contact states of the system. If an identification between states and clusters is found, the SOM will be able to detect errors (a subset of states) in an assembly insertion/extraction task by observing the network response to the on-line measured force signals, i.e., which of the map clusters is more active, and thus, which type of contact state is occurring.

The response of neuron i for a particular input vector depends on the distance from the unit's weight vector to the input sample. For visualization purposes, it is interesting to choose an output function which reaches the maximum when this distance is zero, and decreases monotonically as the distance grows, e. g. a Gaussian:

$$o_i = \exp\left(-\frac{|w_i - x|}{2\sigma^2}\right) \tag{4.1}$$

where:

- o_i is the output, a real value that is represented in the figures with gray levels, white for 1 (maximum activation), and black for 0 (minimum activation)
- w_i is the weight vector
- x is the input vector
- σ is a parameter (the variance) which controls the spread of the function: the smaller it is, the closer the input must be to the weight vector to produce a significant response.

The same input vector is presented to all the neurons. The computation of the neuron responses results in characteristic gray-level patterns. In this patterns, the state associated with the given input can be readily identified as the brighter region on the net.

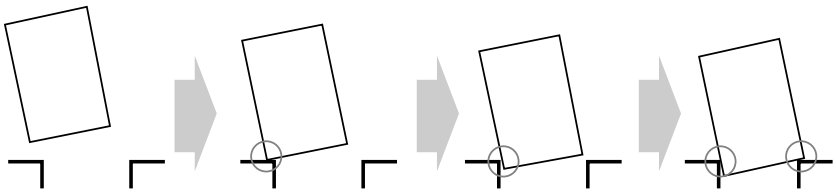


Figure 4.1: Sequence of contacts during a peg-in-hole insertion task

We can also visualize the region activated by a whole sequence of inputs, instead of just a single input. To do so, we just add the activation for each instant, obtaining the resulting pattern for the sequence. These regions can be reduced to individual cells if only the one giving the maximum response is considered. This neuron is usually known as the winner. This concept is useful when monitoring a sequence of signals, since it allows us to visualize the sequence on the map as the trace of the winner cells for each input.

To monitor a plan, and detect an error in the execution, the activity of the network for the measured inputs is computed, and the process state can be readily obtained by observing the winner cell, or the brighter region on the map.

4.3 The peg-in-hole insertion task

In the two-dimensional peg-in-hole insertion task, a rectangular peg has to be inserted into a vertical rectangular hole. Even small errors in the position and/or orientation of the peg prevent the task from being accomplished successfully, causing undesired contacts (see Fig. 4.1) which, if ignored, produce reaction forces which can damage the piece or the manipulator.

In the considered setup, the hole is chamferless, but there is a clearance between the hole and the peg. In the first simulations friction is neglected, but it will be considered later. Force sensing is achieved by means of a simulated force sensor attached to the upper face of the peg. When there is a contact between the peg and the surface, the reaction forces are measured by this hypothetical sensor. As we can see in Fig. 4.2, there are six different possible *contact states* between the peg and the surface of the hole. A contact state is the set of all configurations of contacts between the same topological elements (edges, vertices). By considering some clearance between the peg and the hole we have added three more states to the ones used by Asada [1993]. Obviously, each state has its own symmetric. Our aim is to train a neural network with the torque and forces measured by the sensor in each type of contact, and obtain an output from the network suitable for

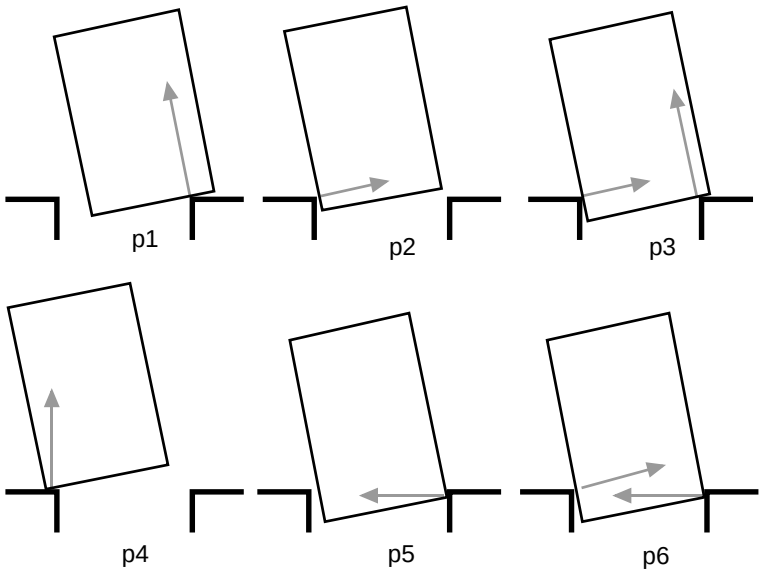


Figure 4.2: Contacts in a peg-in-hole insertion task

identifying these states [Cervera et al., 1995b].

4.3.1 Frictionless simulation

Relative positions between the peg and the hole are randomly chosen, and the appropriate reaction forces and torque are calculated. The peg is tilted up to 15 degrees from the vertical axis in each direction. The *virtual* force sensor is located in the middle of the top edge of the peg. Since only the orientation of the force vector depends on the surfaces in contact, the vectors are normalized to unit length. These are the inputs to the neural network. The height of the peg is twice its width. The clearance of the hole is 1%.

States p1-3

First only states p1, p2 and p3 (and their symmetric ones) are considered (as in [Asada, 1993]). A two-dimensional SOM of 16×9 units hexagonally connected is chosen, with the *bubble* neighborhood function (3.3). Following the recommendations of Kohonen [1995] (page 80), the map is trained in two phases:

1. Ordering. Fairly wide neighborhood and learning rate: 1,000 samples, $\eta = 0.05$ and $r_t = 15$ (3.2 and 3.3).

2. Fine tuning. Small values of neighborhood and learning rate: 10,000 samples, $\eta = 0.01$ and $r_t = 7$.

During each phase, the learning rate and the neighborhood radius are linearly decreased to zero. The resulting mean quantization error was 0.094.

After training, the clusters are identified with an analysis of the output (4.1, with $\sigma = 0.5$) of the network when presented with the inputs of each contact state. These organization of clusters is shown in Fig. 4.3, and represent the mean activation of the map (black is 0, white is 1) for samples of each state (since the samples were randomly chosen, approximately one sixth of the 10,000 samples of the training set correspond to each state). By introducing a threshold τ, $0 < \tau < 1$, the map can be partitioned into regions: all the units whose mean activation is greater than the threshold are labeled as identifying that state.

The regions in this way defined with $\tau = 0.7$ are represented in the map depicted in Fig. 4.4. Each state is identified by a cluster of units with the same color. Units represented by white-filled circles are overlapping units. All the clusters are properly arranged on the map without almost overlapping: only states p1 and p1* are slightly overlapping due to the smooth transition between these states when the peg is normal to the surface. Choosing a lower τ might overlap more states, and a greater one would shrink the regions; for example, the map built from the same activation clusters, but with $\tau = 0.5$ is depicted in Fig. 4.5, resulting in a great deal of overlapping units.

The map is symmetric with respect to the states, which is not surprising due to the physical symmetry of the system. It must be noted that the layout of clusters, and the clusters themselves, have been found by the network without any supervision, only with random samples of the signals. The supervisor only labels the clusters with the identifier of the state. Other simulations produced vertically and horizontally mirrored versions of the same map, but the relative distribution of the clusters remained constant.

States p1-4

If we consider states p1, p2, p3 and p4, we obtain the clusters shown in Fig. 4.6. One can see that clusters for p1 and p4* are located almost on the same units of the map, and the same happens with p1* and p4. This is not a limitation of this particular map, since training with other parameters does not alleviate the problem. The ambiguity is a result of these states sharing the same region of the input space, i.e. similar input signals.

As depicted in Fig. 4.2, the reaction force in state p1 is parallel to the local vertical axis of the peg, while the force in state p4 is parallel to a vertical

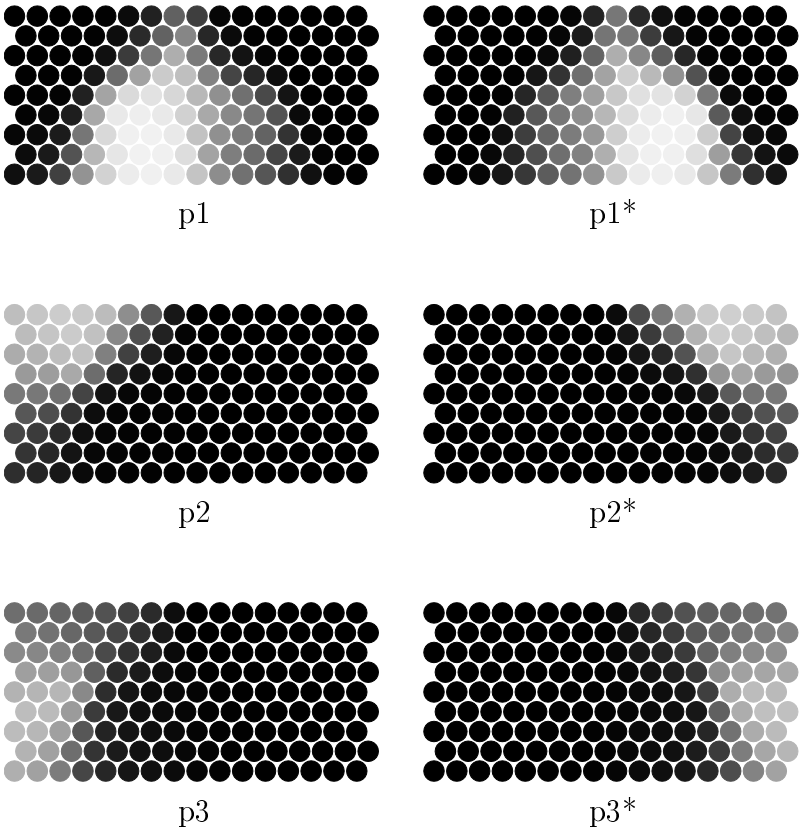


Figure 4.3: Cell activities for a SOM trained with force/torque samples of three contact states, $\sigma = 0.5$

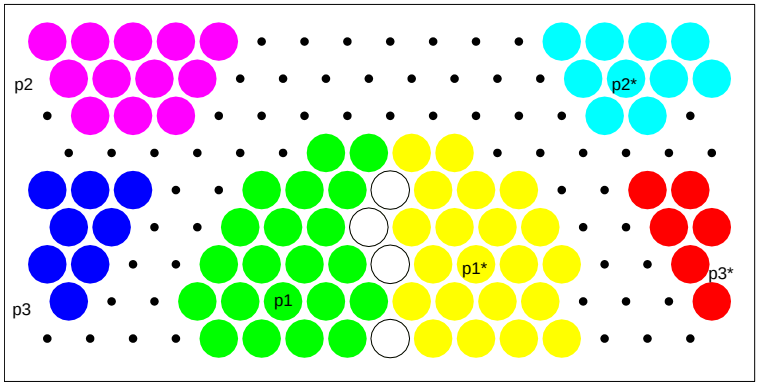


Figure 4.4: Map of regions defined in SOM trained with three contact states, $\tau = 0.7$

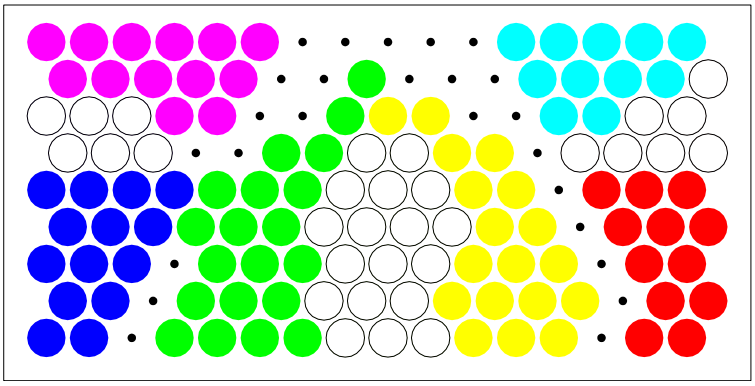


Figure 4.5: Map of regions defined in SOM trained with three contact states, $\tau = 0.5$

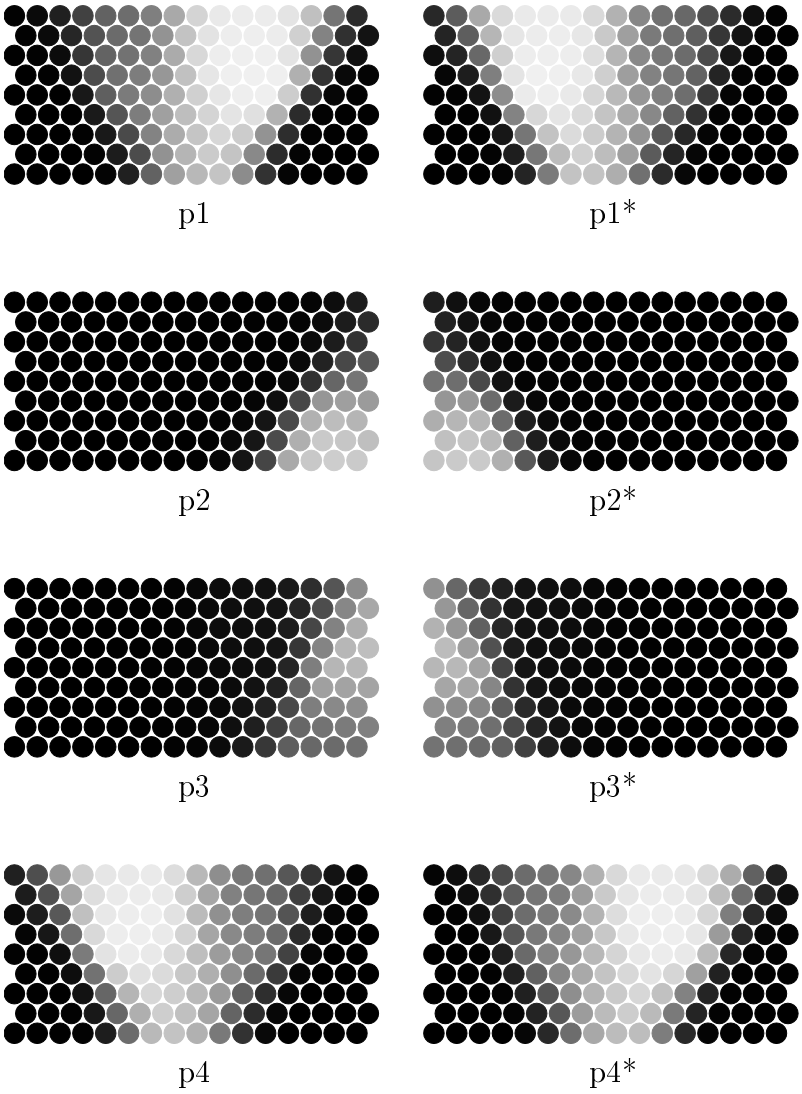


Figure 4.6: Cell activities for a SOM trained with force/torque samples of four contact states, $\sigma = 0.5$

axis relative to the hole. The problem is that the peg is almost vertical with respect to the hole (in our simulations that angle is restricted to a maximum of 15 degrees), and these forces will be very similar, so the sensor response will be similar too. States p1 and p4 can be distinguished by the sign of the torque. But states p1 and p4* have the same torque, and the same happens with p1* and p4.

Consequently these states cannot be uniquely identified if only force and torque information is used. The proposed solution is the use of another available signal, with some different values at each ambiguous state. The signal is treated like any other input to the SOM (with four inputs now), and the whole training procedure is not changed. The only thing is to select the signal, which in this case is the angle of the peg or orientation. The uncertainty of the signal should also be taken into account.

The activity patterns for the new SOM are depicted in Fig. 4.7. Now the overlapping of clusters has been reduced. The resultant map ($\tau = 0.55$) is depicted in Fig. 4.8.

States p1-6

Finally, for all six states, the activation patterns of force/torque signals is shown in Fig. 4.9. Obviously, the number of ambiguities is even greater than before. The following pairs of states are confused: (p1,p4*), (p1*,p4), (p2,p5*), (p2*,p5) and (p6, p6*).

Again, a fourth input is added to the SOM: the angle of the peg. After training with the new values, the activation patterns become more distinguishable (Fig. 4.10) and a map is constructed (Fig. 4.11).

4.3.2 Considering friction

The effect of introducing friction [Cervera and del Pobil, 1996] is that reaction forces are no longer normal to the surface of contact, but, depending on the applied force, might vary within a range defined by the friction coefficient μ. As a result, the distribution of forces for each state spreads over the force space, and is likely to overlap with other states, thus reducing the ability to uniquely identify each state.

In the experiments with friction, the appropriate forces are chosen from the friction cones with a uniform random probability. For the sake of simplicity, the peg angle is kept positive or zero. In the previous section it was shown that negative angles cause ambiguities among states that cannot be solved with only force measurements.

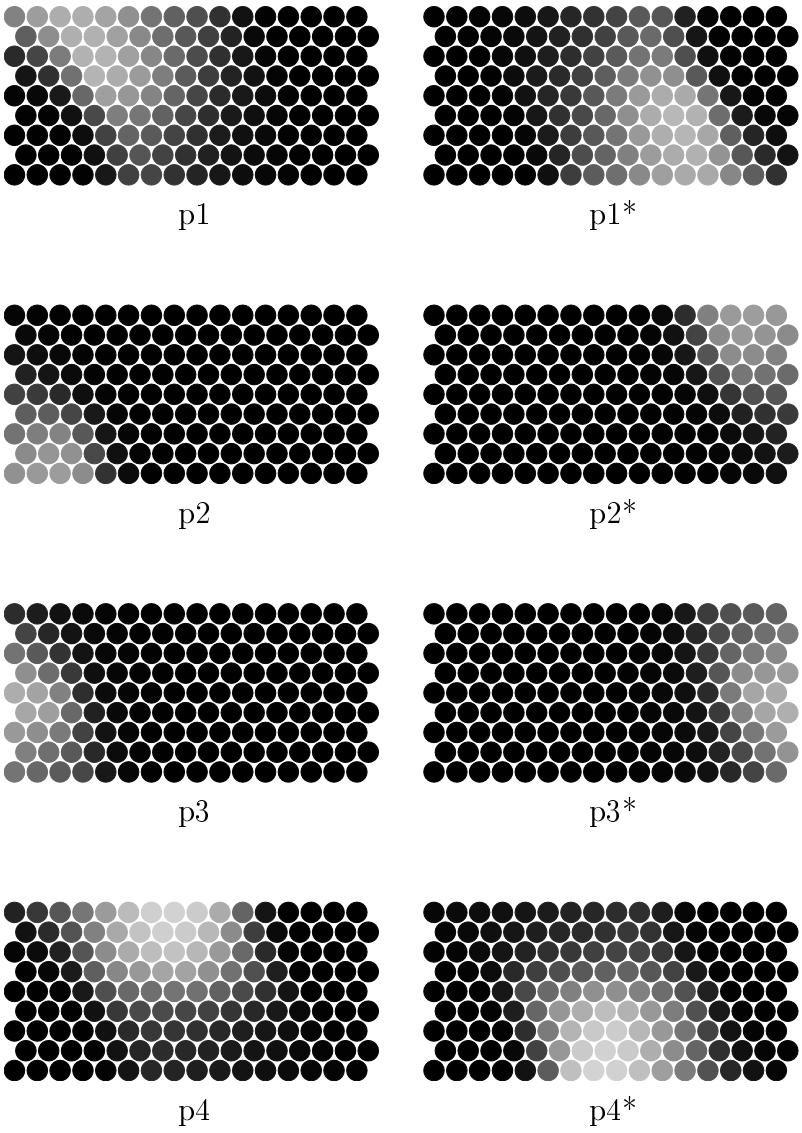


Figure 4.7: Cell activities for a SOM trained with force/torque and angle samples of four contact states, $\sigma = 0.5$

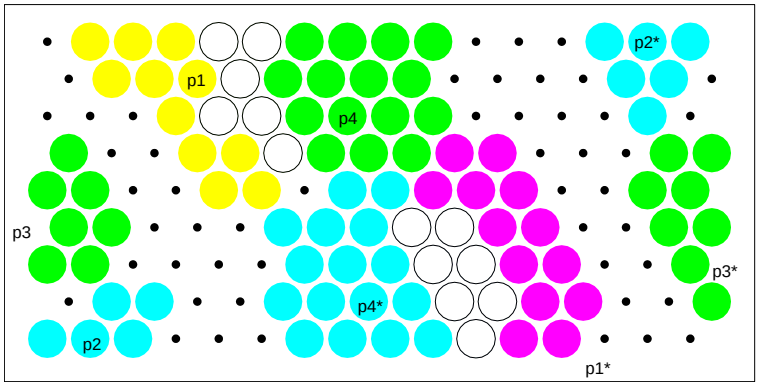


Figure 4.8: Map of regions defined in SOM trained with force/torque and angle samples of four contact states, $\tau = 0.55$

Since we are interested in the influence of the task parameters, and not in those of the network, network dimensions are kept constant (a lattice of 15×10 hexagonally connected units with *bubble* neighborhood) as well as the initial training parameters (learning rate, neighborhood radius, etc.). We investigate the performance of the network for several combinations of clearance and friction (μ) parameters (Fig. 4.12).

The experiments consist of three phases. Each phase involves an independent set of samples which are randomly generated. The same number of samples for each contact state is chosen. Any parameter subject to uncertainty is considered to have a uniform probability density function. In the same way, any random choice is equally probable. Each sample consists of the two force components, normalized to unit module, and the appropriate torque value.

The main difference with the previous experiments is in the calibration phase; a parameter-free calibration scheme is used now, where the number of samples for which the unit wins is what determines the unit's label.

The three phases are the following:

1. Training. The weights are randomly initialized. The neural network is trained with a set of 1,200 random input samples. As before, this process is split into two iterations. The first one (ordering phase) is 3,000 training steps long, and the initial parameters are $\eta = 0.02$ and $r_t = 12$ (Eqs. 3.2 and 3.3). The second iteration (tuning phase) consists

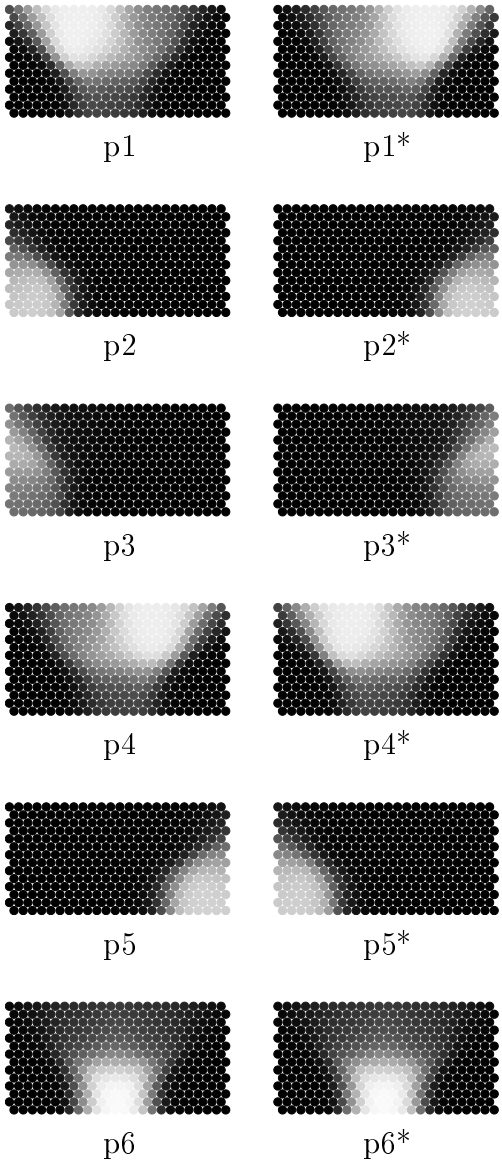


Figure 4.9: Cell activities for a SOM trained with force/torque samples of six contact states $\sigma = 0.5$

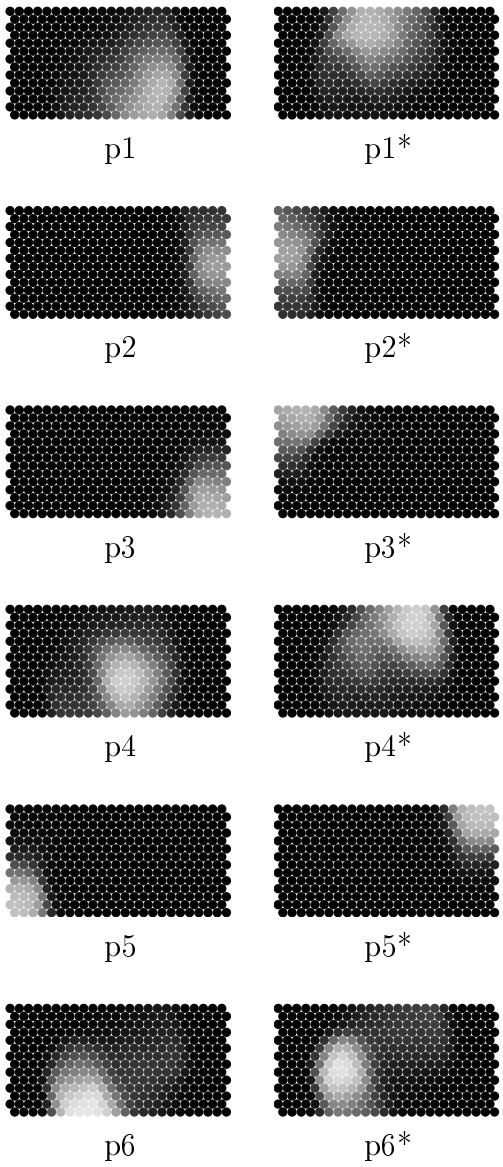


Figure 4.10: Cell activities for a SOM trained with force/torque and angle samples of six contact states $\sigma = 0.5$

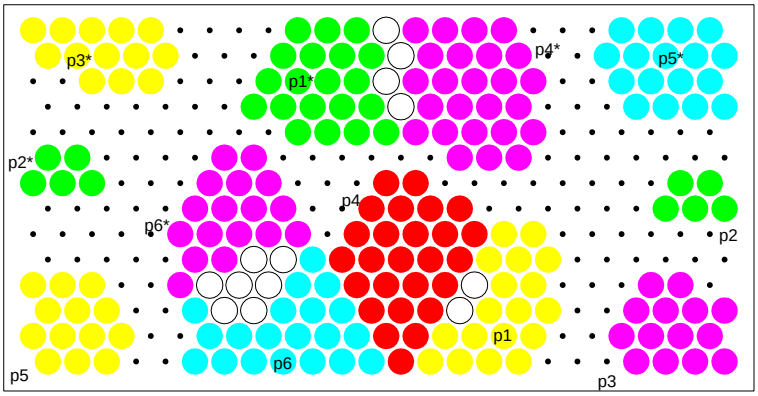


Figure 4.11: Map of regions defined in SOM trained with force/torque and angle samples of six contact states, $\tau = 0.6$

of 60,000 training steps, and initially $\eta = 0.001$ and $r_t = 5$.

2. Calibration. A set of 600 samples is used. The network response is analyzed and state labels are associated with the network units. A unit will be associated to a contact state if that unit's response is greater with input data of that state than with data of any other state. Unlike the previous section, this is calculated by counting how many times a neuron is selected as the closest to the input samples of the different states. The state which collects more *hits* is selected for that unit's label. A second label (of the state with the second number of hits) will also be used during visualization, in order to highlight the overlapping among states in the map. This labeling scheme is somewhat simpler than the used in the previous section, since no additional parameters are needed.

3. Testing. The set consists of 600 samples. The performance of the network is tested with an independent set of data. For each sample, the most responsive unit is selected, i.e. the one whose weights are closer to the input signals. The contact state is simply given by that unit's label. An uncertain response occurs when that unit is unlabeled. In order to solve this problem, we will introduce another method for calculating the network's output.

Results in this section are presented in tables, which contain the per-

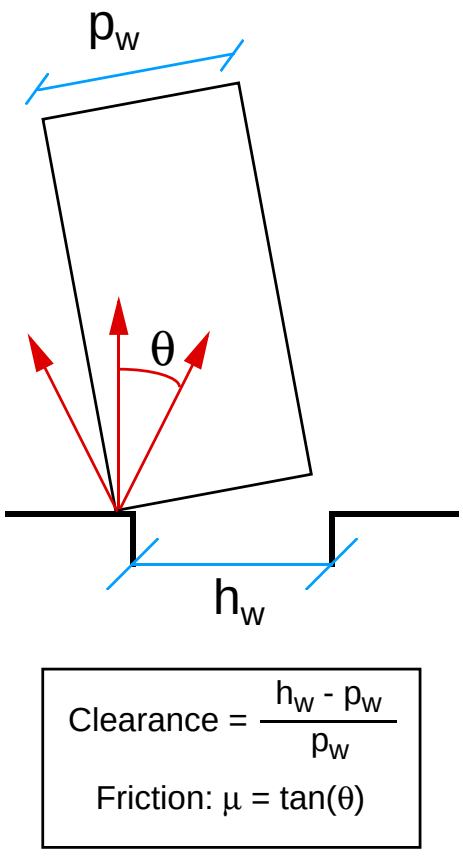


Figure 4.12: Parameters of the peg in hole task: clearance and friction

Table 4.1: Confusion matrix for $\mu = 0.2, Clearance = 1\%$

%	p1	p2	p3	p4	p5	p6
p1	59	–	–	5	–	12
p2	–	100	–	–	–	–
p3	–	–	100	–	–	–
p4	10	–	–	94	–	9
p5	–	–	–	–	97	–
p6	24	–	–	–	–	78
None	7	–	–	1	3	1

centages of classification for the contact states. A table can be considered a *confusion matrix*, since each column holds the distribution of the network response for each state. A perfect identification would result in all diagonal values near 100% and the rest of matrix values near 0.

Results for a experiment with $\mu = 0.2$ and $clearance = 1\%$ are shown in Table 4.1. Two states, p2 and p3, are perfectly identified, according to the test set. Other two, p4 and p5, are almost perfectly identified. State p4 is correctly identified in the 94% of the cases, it is erroneously identified as p1 in the 5% and it is unclassified in the remaining 1%. Meanwhile, p5 is properly classified in the 97% of the cases, but it is unknown in the remaining 3%. The other two states, p1 and p6, are more ambiguous, and the proper classification percentages are smaller. The average network performance is very good, a 88% success, and we must take into account that only force/torque information has been used.

Ambiguities are minimized since the symmetric states have not been considered. In the previous section, confusion arose between states with different orientation, thus needing to include this signal in the network.

An alternative method for representing the network is the so called u-matrix visualization [Ultsch, 1993]. This display method has also been described in [Kraaijveld et al., 1992], and consists in visualizing the distances between reference vectors of neighboring map units using gray levels. The farther the distance, the darker the representation. In this way, one can identify clusters, or groups of neurons with similar response, which should be desirable to belong to the same contact state.

The map is represented in Fig. 4.13 , which consists of a big white region on the top with units labeled with states p1, p4 and p6, and three smaller light regions isolated by darker zones, i.e. long distances. This regions are labeled with p2, p3 and p5. This representation reflects the state ambiguities, which are also presented in the table. Some units are labeled twice to show

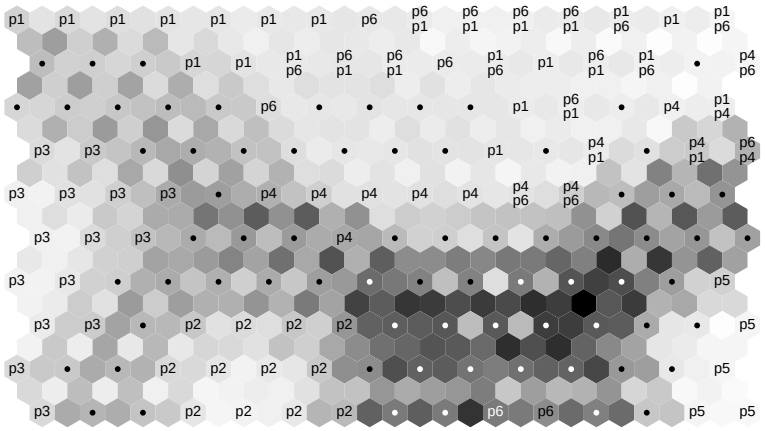


Figure 4.13: U-matrix representation of the neural network. Task parameters: $\mu = 0.2$, Clearance = 1%

this problem, that occurs with states p1, p4 and p6. This means that those units not only are selected for the first state, but sometimes they are also selected for another state. Unlabeled neurons are displayed as a dot.

It should be recalled from the previous section that, if additional states are considered, the network is unable to improve unless more information is available by adding new input signals. These could be positional or other kind of useful information. However, tactile information is sufficient to get a good identification of the contact state, according to the classification rate which is obtained in the experiments.

4.3.3 Robustness against task changes

We have argued that an accurate task model is useless if the task parameters change unpredictably. We want to test our neural network against these changing conditions. The same network that was trained in the previous experiment is now tested with input data which has been generated with other parameters, namely a different clearance or friction.

First, the clearance is decreased to 0.2% and 0.5%. Without further training of the network, its performance is maintained or even improves a little bit (see Tables 4.2 and 4.3). This could be due to the fact that it is easier to identify contacts when the clearance is small. The global classification rates are 89% right, 8.5% wrong, and 2.5% unknown, for a clearance of 0.5%, and 89.3% right, 7.3% wrong, and 3.3% unknown for a clearance of 0.2%.

Secondly, the clearance is increased up to 2%. The network's performance is only slightly worse: 84.7% right, 11% wrong, and 4.3% unknown

Table 4.2: Confusion matrix for $\mu = 0.2, Clearance = 0.2\%$

%	p1	p2	p3	p4	p5	p6
p1	65	–	–	2	–	11
p2	–	100	–	–	–	–
p3	–	–	98	–	–	–
p4	6	–	–	96	–	7
p5	–	–	–	–	95	–
p6	21	–	–	2	–	82
None	8	–	2	–	5	–

Table 4.3: Confusion matrix for $\mu = 0.2, Clearance = 0.5\%$

%	p1	p2	p3	p4	p5	p6
p1	61	–	–	1	–	14
p2	–	100	–	–	–	–
p3	–	–	96	–	–	–
p4	6	–	–	98	–	3
p5	–	–	–	–	97	–
p6	26	–	–	1	–	82
None	7	–	4	–	3	1

(Table 4.4). Consequently, the self-organizing map is robust against clearance changes (Fig. 4.14 depicts the relationship between clearance and the percentage of correct classifications).

Next we consider some variations in the friction coefficient. If this coefficient is halved ($\mu = 0.1$), the network's performance improves up to 90.1% right, 8.1% wrong, and 1.7% unknown (See Table 4.5). The reason is that friction cones are narrower and they are more unlikely to overlap and to cause ambiguities. However, if the coefficient is increased two and four times, performance progressively worsens down: 74.8% right, 11.5% wrong, and 13.7% unknown ($\mu = 0.4$); 54% right, 12.3% wrong, and 33.7% unknown (see Tables 4.6 and 4.7, and Fig. 4.15).

It is interesting to note that the network's response gracefully degrades as the friction coefficient increases. The unknown value indicates that the network is doing a good job in the sense that states are not randomly misclassified. Instead, the network might issue a warning to a higher control level, indicating that there is something wrong the network cannot cope with and further training or redesigning is required.

Finally, the network also generalizes to a combination of changes of clear-

Table 4.4: Confusion matrix for $\mu = 0.2, Clearance = 2\%$

%	p1	p2	p3	p4	p5	p6
p1	53	–	–	1	–	19
p2	–	100	–	–	–	–
p3	–	–	100	–	–	–
p4	13	–	–	98	–	7
p5	–	–	–	–	93	–
p6	26	–	–	–	–	64
None	8	–	–	1	7	10

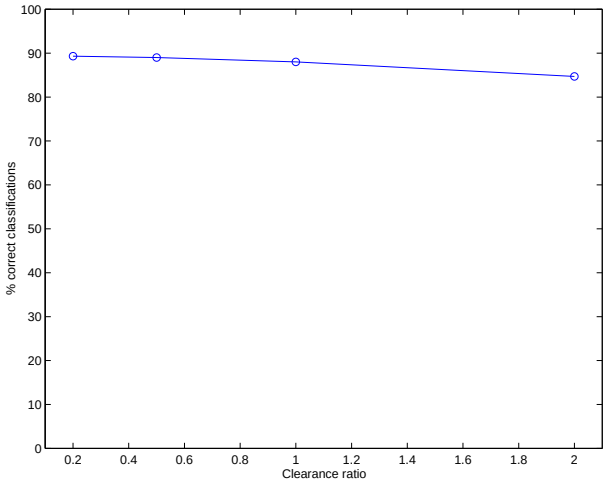


Figure 4.14: Evolution of the percentage of correct classifications with respect to the changes in the clearance ratio ($\mu = 0.2$). Network was trained with $Clearance = 1\%$

Table 4.5: Confusion matrix for $\mu = 0.1, Clearance = 1\%$

%	p1	p2	p3	p4	p5	p6
p1	55	–	–	2	–	5
p2	–	100	–	–	–	–
p3	–	–	99	–	–	–
p4	7	–	–	98	–	4
p5	–	–	–	–	100	–
p6	31	–	–	–	–	89
None	7	–	1	–	–	2

Table 4.6: Confusion matrix for $\mu = 0.4, Clearance = 1\%$

%	p1	p2	p3	p4	p5	p6
p1	42	–	–	2	–	7
p2	–	95	–	–	–	–
p3	–	–	90	–	–	–
p4	11	–	–	84	–	10
p5	–	–	–	–	64	–
p6	30	–	–	9	–	74
None	17	5	10	5	36	9

Table 4.7: Confusion matrix for $\mu = 0.8, Clearance = 1\%$

%	p1	p2	p3	p4	p5	p6
p1	24	–	1	2	–	11
p2	–	74	–	–	–	–
p3	6	–	70	–	–	–
p4	13	–	–	57	–	5
p5	–	–	–	–	37	–
p6	18	–	1	17	–	62
None	39	26	28	24	63	22

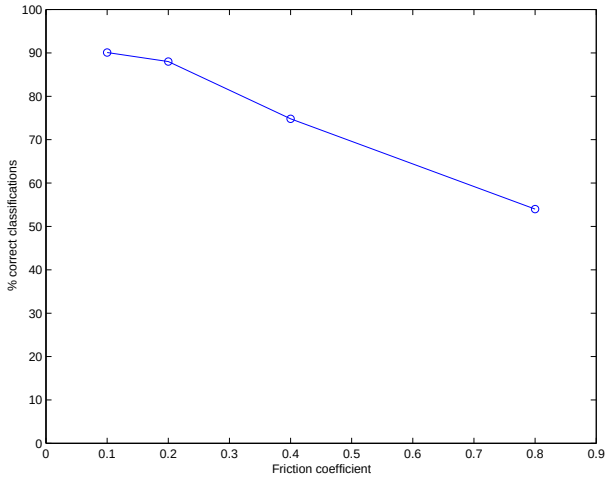


Figure 4.15: Evolution of the percentage of correct classifications with respect to the changes in the friction coefficient ($Clearance = 1\%$)

Table 4.8: Confusion matrix for $\mu = 0.4, Clearance = 2\%$

%	p1	p2	p3	p4	p5	p6
p1	31	–	–	1	–	10
p2	–	90	–	–	–	–
p3	–	–	87	–	–	–
p4	12	–	–	87	–	11
p5	–	–	–	–	59	–
p6	39	–	–	7	–	76
None	18	10	13	5	41	3

ance and friction coefficient ($\mu = 0.4$, Clearance = 2%). Classification rate is slightly lower, 71.7% right, 13.3% wrong, and 15% unknown (Table 4.8).

4.3.4 Adaptation to permanent changes

It has been shown how the network gracefully degrades under task parameter changes. However, if the performance falls under a certain level, the output will be absolutely useless, or even harmful if the contact states are confused. Our neural network tends to be conservative, and outputs an unknown state in this case, but results show how misclassification of wrong states grows up too.

Considering the difficult problem with a greater friction coefficient, we train a new neural network to demonstrate that the task is still solvable. Results are shown in Table 4.9, and they are a great improvement over the old network, because training, calibrating and testing have been carried out with data sets generated with the large friction coefficient. Nevertheless, performance is a bit lower than the first problem because the task is more difficult. Now, friction cones are wider and are more likely to overlap and to cause ambiguities between contact states. The successful classification rate is 77.5%, more than 23% better than the network in Table 4.7.

Frequently, some tasks do not allow to stop the system and train a new neural network in order to adapt to the new conditions. A desirable system should be capable of adapt on-line, while the process is running, to these changes. We demonstrate the ability of our neural network to adapt in this way. The first network is re-trained, i.e. its weights are not initialized randomly but are kept as the initial weights and another training iteration is performed. This process could be done on-line if the training data are collected from the running process.

After 20,000 training steps, the network's performance is 77% right, 19.8%

Table 4.9: Confusion matrix for a new SOM trained with $\mu = 0.8, Clearance = 1\%$

%	p1	p2	p3	p4	p5	p6
p1	53	–	15	3	–	41
p2	–	95	–	4	–	–
p3	15	–	82	–	–	1
p4	20	2	–	93	–	12
p5	–	–	–	–	96	–
p6	6	–	–	–	–	46
None	6	3	3	–	4	–

Table 4.10: Confusion matrix for adapted SOM, retrained with $\mu = 0.8$, 20,000 steps, $learning \quad rate = 0.001$ and $radius = 5$

%	p1	p2	p3	p4	p5	p6
p1	40	–	11	2	–	23
p2	–	97	–	2	–	–
p3	16	–	81	–	–	5
p4	21	1	–	95	–	21
p5	4	–	–	–	100	–
p6	11	–	1	1	–	49
None	8	2	7	–	–	2

wrong, and 3.2% unknown (see Table 4.10). After 60,000 training steps, the performance is 78.5% right, 19.2% wrong, and 2.3% unknown (see Fig. 4.16). It is even better than the network trained from scratch with new data. This is a very important result that demonstrates the ability of the network to evolve under new task conditions. It is also an example of learning a difficult task from an initial easier task. The network was first trained with data generated with low friction coefficient, which makes contact classification easier. Afterwards, the network is re-trained with data from the harder task while keeping the previous knowledge of the similar task. Moreover, this transition is smooth, providing robustness and trustfulness to the system to run on-line in demanding processes like assembly tasks.

4.3.5 Collective output calculation

In the previous experiments, the output of the network was given by the unit whose weights were closest to the input signals. That unit's label was the

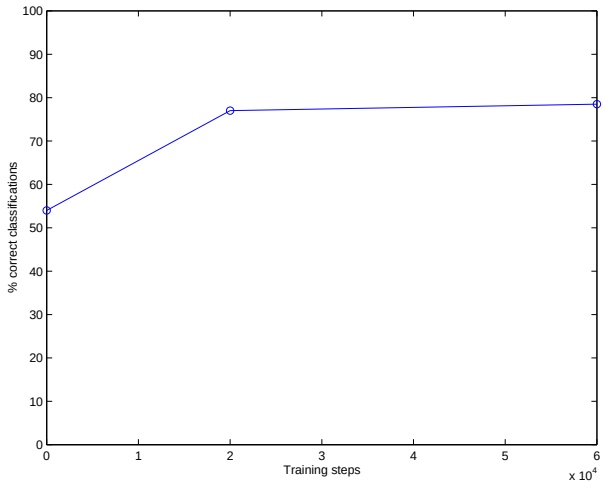


Figure 4.16: Evolution of the percentage of correct classifications during the retraining process

contact state. This method has two minor drawbacks: the first one is the subset of unlabeled units, i.e. those units that never were selected during the calibration phase. Thus, they have no label at all, and the network's output will be unknown, if one of them is selected. This could be overcome if the search is restricted to the labeled units, but that implies wasting resources, since those units were also trained during the training phase. The second problem is that the network does not give any measure of confidence, i.e. some kind of probability of being right. Our new method will overcome these two problems.

In our method, during the calibrating phase a vector of hits is collected for each unit. This hit vector has as many integer components as different contact states. Each component will store the number of times that neuron was selected for that given state during the calibration process. Obviously the maximum component will correspond to the state label that should be assigned by the original procedure. Now, the network's output is calculated with the hit vectors of all the units in the map. The output vector is a weighted sum of these vectors. The coefficients are functions of the distance from the input signal to each of the unit weights, and a negative exponential function is chosen (similar to Eq. 4.1). Thus the response of the map for an input x would be:

$$output = \sum_i \left(hit_i \exp\left(-\frac{|w_i - x|}{2\sigma^2} \right) \right) \tag{4.2}$$

where σ is a coefficient of spread and w_i the weight vector of unit i. The

Table 4.11: Confusion matrix with collective output calculations, $\mu = 0.8$, $\sigma = 0.2$

%	p1	p2	p3	p4	p5	p6
p1	63	–	9	–	–	25
p2	–	99	–	7	–	–
p3	16	–	88	–	–	5
p4	21	1	3	93	–	31
p5	–	–	–	–	100	–
p6	–	–	–	–	–	41

Table 4.12: Output vectors for different inputs $\mu = 0.8$, $\sigma = 0.2$

Input	Output
(-0.09, -0.99, -0.50)	(4.4e-3, 2.3e-7, 5.3e-10, 1.3e-11, 2.7e-9, **15.1**)
(0.16, -0.98, -0.19)	(5.5e-5, 1.4e-4, 2.1e-8, 9.8e-10, 3.1e-9, **22.7**)
(0.31, 0.95, -0.62)	(**12.76**, 0.01, 0.61, **15.25**, 0.03, **10.8**)
(-0.60, -0.79, 0.47)	(1.4e-7, 8.9e-8, 1.8e-11, 7.1e-13, 4.5e-11, **6.9e-3**)

smaller σ, the smaller the influence of the further units. The maximum component of this output vector is selected, and the network's output is the contact state associated to that component. In Table 4.11 classification rates using this method are shown, for $\mu = 0.8$. The unknown column does not appear anymore.

With this method, a state is always selected. The successful classification rate is improved to 80.7%. For $\mu = 0.2$, the successful classification rates are 87.2% and 88.2%, which are very close to the original 88% rate.

Despite this little improvement in the classification rate, the major advantage is the confidence measurement. The magnitude of the vector components can be used for this purpose. If one state is clearly distinguished, its component will be much bigger than the others. However, if there is a confusion between several states, the magnitude of their components will be similar, reflecting this ambiguity. Finally, if no state is confident enough, every component of the output vector will be small. For example, the samples in Table 4.12 were generated for state p6 with $\mu = 0.8$, and the output vectors were calculated with $\sigma = 0.2$.

The outputs to the two first samples are confident. The value of the 6th component is far greater than the others, thus the sample will be classified as state p6. In the third example, however, several components share the same order of magnitude. With the original algorithm, the signal would be

classified as state p4, the biggest component. However, this classification is not confident due to the similar order of magnitude of the first and sixth components; there is an ambiguity among these states. In the fourth example, the sixth component is at least four orders of magnitude greater than the others, but its absolute magnitude is small. The most likely state is p6, but the network is not very confident about that.

The absolute values of the components depend on the parameter σ and the network metrics, which in turn depend on the input signal space metrics. A proper tuning of this parameter is required to maximize the mean classification rate. This is a mostly heuristic procedure.

4.3.6 Discussion

A learning approach to contact classification based on unsupervised neural networks and force sensing has been applied to a simple peg-in-hole task. Despite this simplicity, there are fundamental ambiguities between some states, which are not detected by force sensing and pose the necessity of other signals.

Though SOMs were not specifically designed for classification tasks, their performance in the identification of contact states with friction is excellent. Besides that, the network is robust against changes in task parameters (clearance, friction), and it behaves in a gracefully degrading way, providing a highly confident and reliable output for demanding robot assembly tasks. The neural network is also capable of adapting to these new conditions by means of an on-line retraining process. The performance of the network increases up to the level of those which are specifically trained. In this way, the network is able to adapt to the new conditions of a harder task, by using the knowledge that it previously learned in a simpler problem.

The effect of friction is particularly important for the correct identification of contact states. In the next chapter, it will be taken into account to minimize its effects, since it can slow or prevent the agent from learning the task.

The method of collective output calculation is a new procedure which determines the state in a robust manner, since it not only relies on the winner units but on the activity of the whole map. It provides a method for identifying samples for which the winner unit has not been labeled during the training process.

This section has demonstrated the feasibility of using SOMs for identifying contacts in fine motion tasks. The SOM is able to approximate the distribution of the input signals with the subspace of units, and this approximation captures the main features needed for the identification of contact

states.

In the next section, experiments on detecting contacts in a different setup are carried out. Later on, a simple insertion plan is introduced. The intention of these experiments is to pave the way for the development of the complete autonomous learning architecture in the next chapter.

4.4 A flexible manufacturing system task

The second application of SOMs in contact identification will take place in the context of an actual flexible manufacturing system (see Figs. 4.17 and 4.18). In this system, there is a machining center which works with several types of tools (Fig. 4.19). These tools are fed into the unit by a robot arm (ABB IRB 2000), which also picks the tools from a robot vehicle (Fig. 4.20). The tools are carried from this robot to the machining center and vice-versa. The robot arm is very dependent on the spatial positions of the tools. A small displacement of the vehicle can led to a deficient grasp or ungrasp operation of the tool, which could even be dropped by the arm.

Obviously, the greater source of error is the control of the vehicle. An additional amount of uncertainty is accrued in every displacement. If the vehicle is an AGV that follows a line on the floor and landmarks at the stop points, the error is smaller than for a free navigation vehicle. In our system the robot vehicle is a Robosoft Robuter for free navigation. Although the positioning uncertainty is being improved with the use of ultrasonic sensing, errors of a few millimeters may be expected after several trips. The position uncertainty of the arm can also led to an incorrect insertion or extraction of the tool in the machining center.

On top of the vehicle a loading/unloading unit for pallets is placed. The tool pallet slides into this unit and can be carried by the vehicle. The pallet can hold six tools, for each one there is a clamping device with two flexible claws (Fig. 4.19). The two parallel jaws of the robot arm move a special gripper with the shape of the tools (Fig. 4.21).

In order to detect all these error conditions, a wrist force/torque sensor is attached to the robot arm. The sensor measures the three components of the force and torque vectors acting on the end-effector. Monitoring the force and torque signals should help us detect the error conditions which lead to incorrect insertions or extractions. Moreover, they should be detected while it is still possible to recover from the error. Our aim is to develop a monitoring system with neural networks. It should learn from examples of the task, and be able to distinguish the contact states between the arm and its environment in order to detect good and bad insertions at an early enough

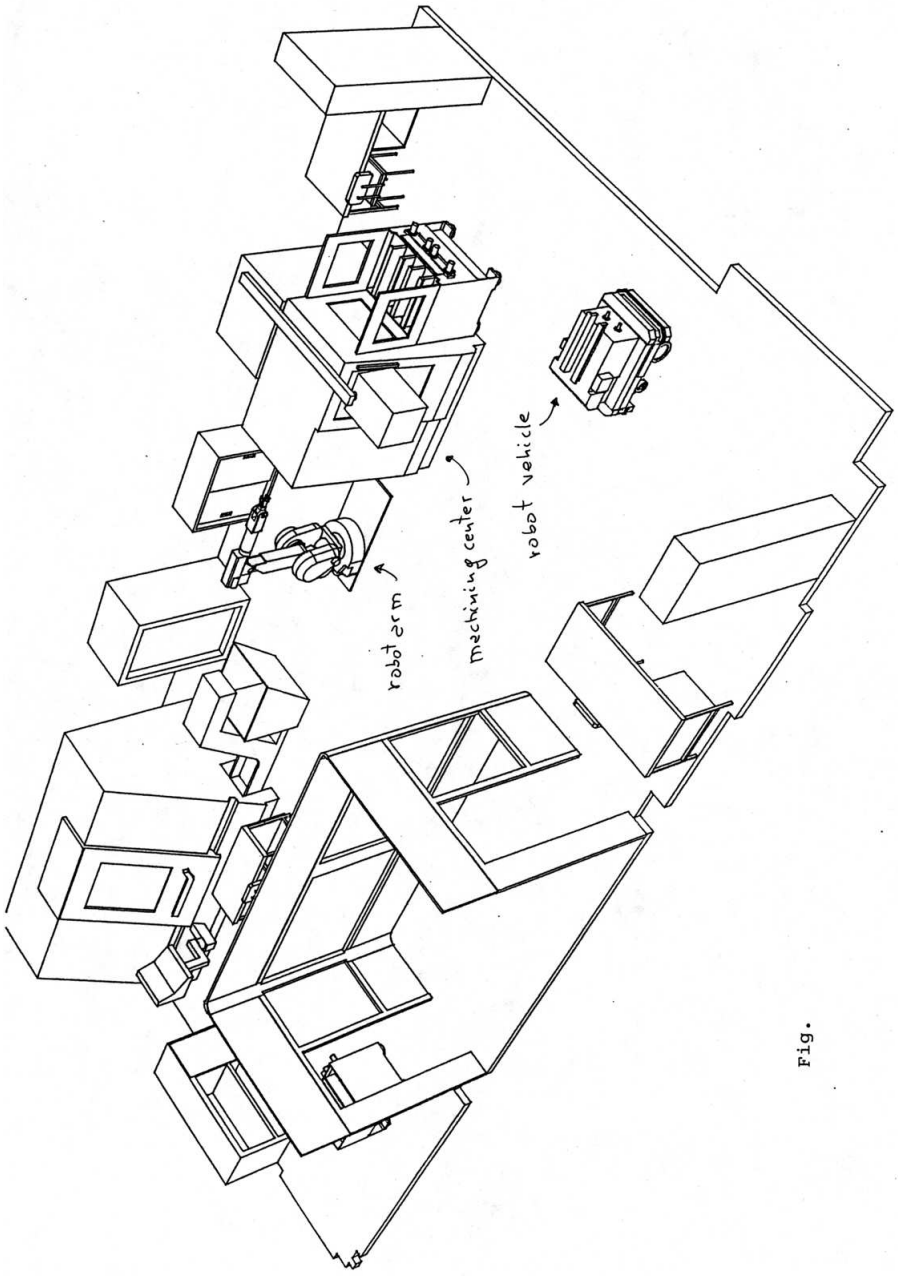


Figure 4.17: CAD representation of the flexible manufacturing system

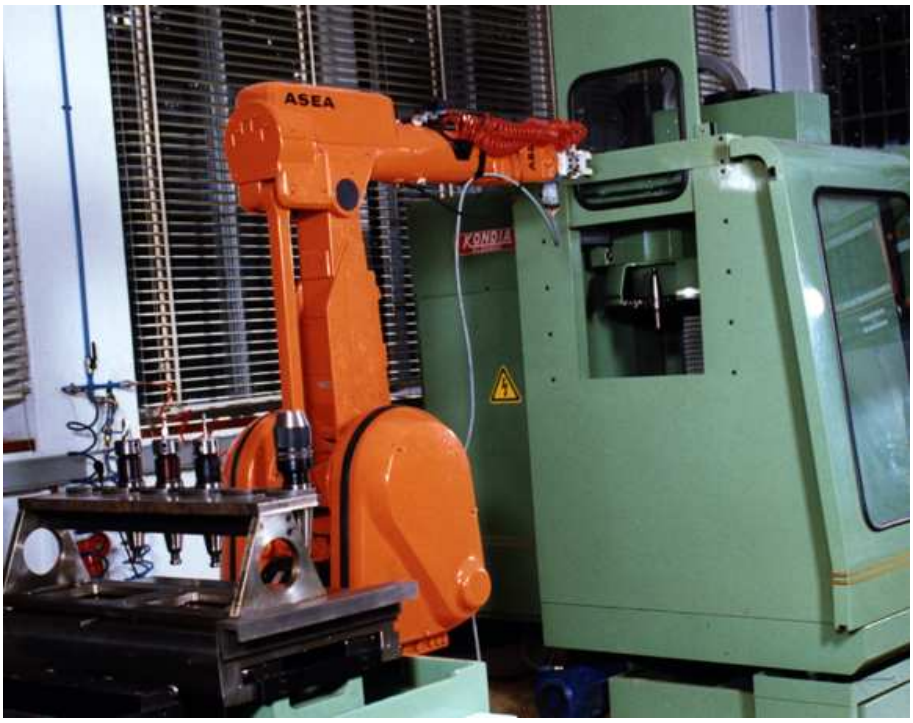


Figure 4.18: View of the tool pallet, robot arm, and machining center

Figure 4.19: View of the pallet carrying different tools

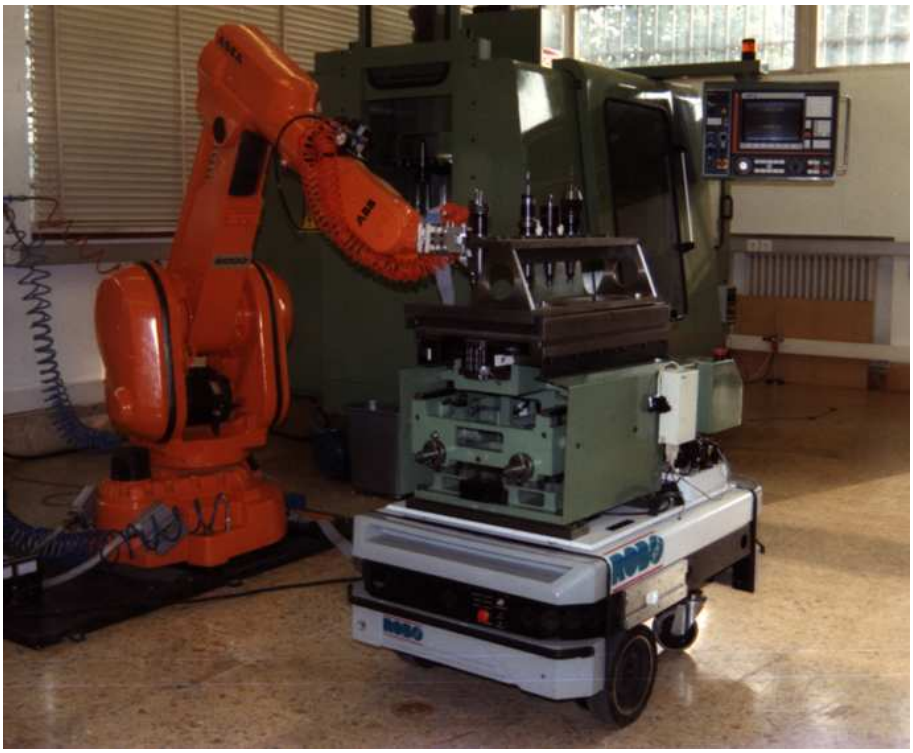


Figure 4.20: Robot arm and mobile vehicle

Figure 4.21: Gripper of the robot arm

stage of the process [Cervera et al., 1996].

It is worth noting that the complexity of these actual operations and similar manufacturing tasks is considerably greater than the usual test-cases in geometric approaches. First, it is a 3D problem with a non-trivial geometry due to the shape of the tool and the insertion place. Second, due to the existence of two flexible claws in the clamping device, the arm must exert a force that makes them yield for the tool to be properly inserted or extracted (see Fig. 4.19).

4.4.1 Learning complex insertion tasks

It has been shown how a self-organizing network can evolve to form clusters closely related to contact states, without any a-priori knowledge of those states. We have solved the problem in the case of the peg-in-hole. Now we want to apply this scheme to the real situation described. The neural network will be fed with the six signals of a real force/torque sensor attached to the wrist of the robot arm. We will limit the analysis to the fine motion involved in the tasks of inserting the tool in the pallet on the robot vehicle or the machining center. A similar treatment could be done for the task of

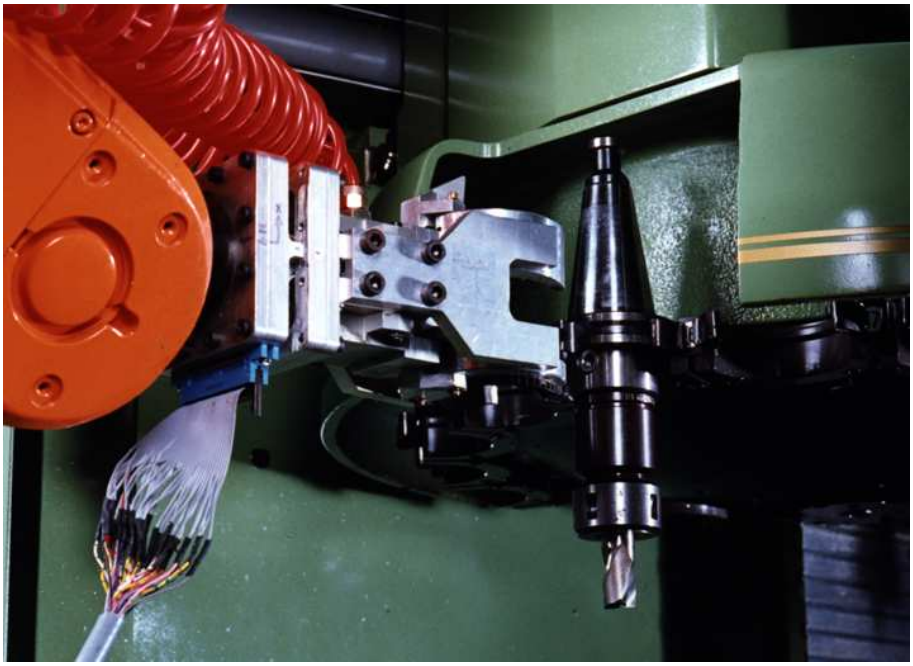


Figure 4.22: Insertion of tool in the machining center

extracting the tool.

4.4.2 Maps for a complete insertion operation

Insertion in the machining center with positive errors

In the first experiment, the tool is inserted in the tool-exchange device at the machining center shown in Fig. 4.22. The uncertainty has been simulated by introducing small position errors in the arm. The network was trained with samples of 100 sensor signals for each insertion operation. Only four situations were considered: offsets of 0 mm (correct insertion), +1, +2 and +3 mm (incorrect insertions) on OY axis (left-right) for the wrist. Twenty-four insertions were carried out (six for each offset). In Fig. 4.23 several examples of sensor data are shown. The differences are clearly apparent from the sensor signals, but an interpretation of these signals is rather difficult.

The SOM consists of $14 \times 8 = 112$ units, hexagonally connected, with *bubble* neighborhood, initialized with random values. Training consisted of the two usual phases: the ordering phase, with 4,000 training steps, initial $\eta = 0.2$ and radius $r_t = 8$; and the fine-tuning, with 40,000 steps, initial $\eta = 0.04$ and radius $r_t = 3$. Force and torque samples were normalized to unit length.

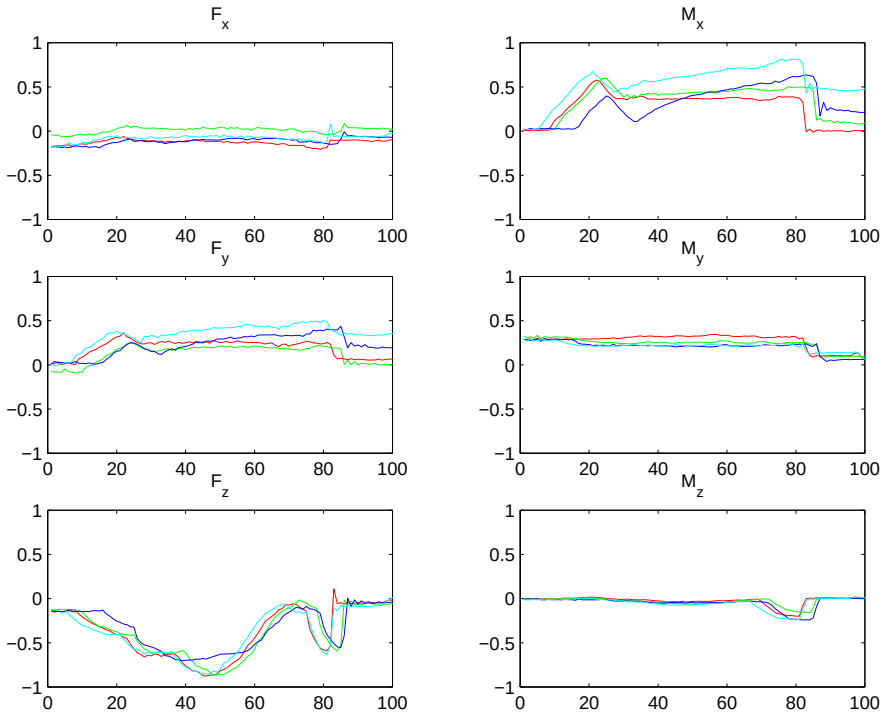


Figure 4.23: Force sensor signals for the insertion task in the machining center with offsets 0, +1, +2, and +3 mm

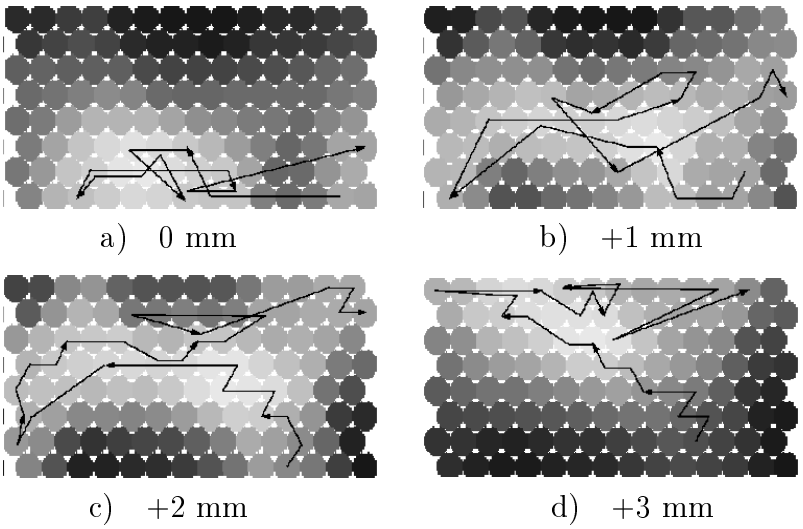
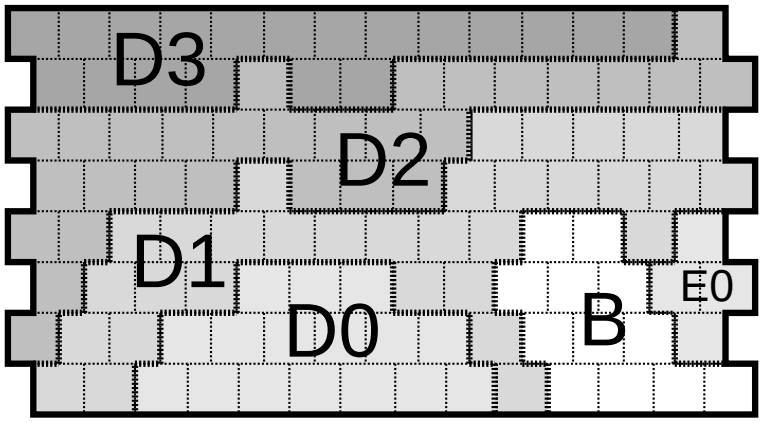


Figure 4.24: Activation patterns ($\sigma = 0.5$) and trajectories of winner units for the insertion task in the machining center with positive transversal error

After training the network with data from a set of examples, the regions on the map are labeled appropriately. In this case, a label is assigned to each different error/offset. In Fig. 4.24 the activation patterns for complete insertion sequences are shown. Each pattern correspond to a different error. In each pattern, the brighter cells are the ones that were more activated in some moment of the process. On the figures, trajectories are also shown, which depict the temporal evolution of the activation pattern. The arrows join the winner units during a whole sequence of signals.

Figure 4.24(a) depicts the cell activation for the correct insertion with no error. The most active cells are located in the lower part of the map. As a greater error is introduced, we can see in the other three plots 4.24(b-d) how active cells are located in the upper zones on the map, in such a way that the greater the error, the upper the most active cells are. This continuity is a consequence of the topological properties of the SOM that preserves neighborhood relations: neighbor cells tend to respond jointly to similar inputs.

Fine motion planning for the insertion task can be monitored to detect errors by means of this network. The resulting map can be partitioned into four regions, according to the activation borders (Fig. 4.25). Since the transition between offsets is continuous, there is an inherent overlapping, which in this case is not harmful (e.g. if the error were exactly 1.5 mm it would be located either at the top of region 1 or at the bottom of region 2). So, instead of choosing a fixed threshold, a unit is labeled with an offset when



B:Common start region
D0,E0:No error
D1:Error=1mm
D2:Error=2mm
D3:Error=3mm

Figure 4.25: Map partition for the insertion task in the machining center with positive transversal error

its activation with samples of such offset is greater than with the others. The absence of unlabeled units might be justified by the smooth distribution of the input signals across the map

The existence of common regions must be taken into account. Prior to the contact with the fixed part, the offset obviously has no effect. An extra region B is added (and, if necessary, a final region E), which corresponds to those samples sensed before the actual contact. This contact is easily detected by a threshold on the normal force (Z).

Each region is labeled with the process corresponding to no error, or +1, +2, or +3 mm errors. Then, we can monitor a new unknown motion for which a positive uncertainty in OY axis for the actual goal position may be expected in the range [0 mm, +3 mm]. The network activation pattern will permit not only to detect an incorrect insertion, but also to identify approximately the magnitude of the error. Indeed, the further the activation pattern from the lower region D0 is and towards the upper region, the greater the positive error will be. A correct insertion will stay between the borders of region D0. Borders should not be seen as precise, a certain overlap exists between regions. The existence of two additional regions B and E0 can be accounted for as follows. B corresponds to a common starting zone, since the sensor signals for the insertion tasks have been taken a few instants before actual contact. Similarly, there is a final ending zone E0 because the ungrasp

action is included in the monitored process. E0 could be extended to include the ending areas for cases D1, D2 and D3.

Insertion in the tool pallet with positive and negative errors

In the real world, errors can obviously occur in either direction. Both positive and negative offsets are considered now, but only for the OY (left-right) axis. The other two axes are not so important: errors in OZ (front-back) only affect the distance to contact; instead of moving a fixed amount towards the goal, a guarded motion automatically detects the contact, regardless the unknown offset. Errors in OX (up-down) are expected to be much smaller, since the main source of uncertainty is the mobile vehicle, and it moves across the floor, but its height (i.e. the vertical position of the tool pallet) does not change. Errors in the orientation of the goal are not explicitly considered in this setup.

To perform these experiments, the tool-pallet has been removed from the robot vehicle and placed in a fixed position (see Fig. 4.19). The number of different possible situations that the network must recognize has been increased by considering errors in the range [-4 mm, +4 mm] on OY axis (in the direction along tool centers on the pallet). To train the network, we have used a set of sensor signals corresponding to insertion tasks with offsets of -4, -3, -2, -1, 0, +1, +3 and +4 mm. Each insertion sequence is made of 55 sample signals.

The SOM consists of $11 \times 9 = 99$ units, hexagonally connected, with *bubble* neighborhood, initialized with random values. Training consisted of the two usual phases: the ordering phase, with 4,000 training steps, initial $\eta = 0.2$ and radius $r_t = 9$; and the fine-tuning, with 40,000 steps, initial $\eta = 0.04$ and radius $r_t = 3$. Force and torque samples were normalized to unit length.

Once the network has been trained, the activation pattern for the complete sequence in each case is obtained (with the same procedure used in the previous experiment). The patterns, and trajectories of winner cells, are depicted in Fig. 4.26; it is more complex than in the previous case, since more states have been introduced. The correct insertion is approximately located along the diagonal (left-down to right-up) of the map, the processes with positive errors move towards the upper left corner the greater the error is, while the negative errors give rise to regions that move towards the lower right corner. This is an effect of the strong symmetry of the task.

Figure 4.27 shows the partitioned map with labeled regions. In two cases, a couple of states were merged in the same region, since the resulting activation patterns presented a great overlapping. This fact can be accounted

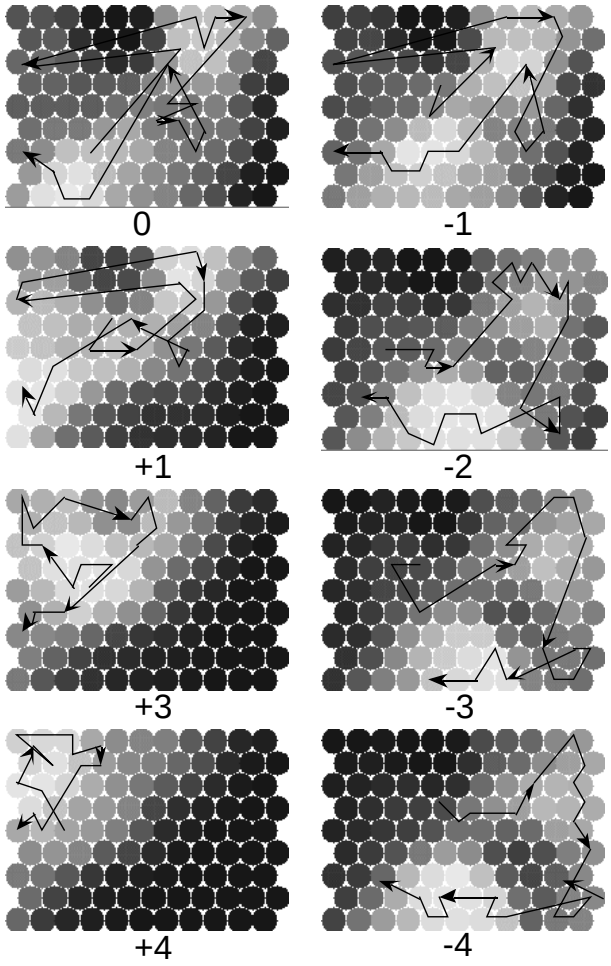


Figure 4.26: Map activations and traces of winner units for the insertion task in the tool pallet with positive and negative transversal error

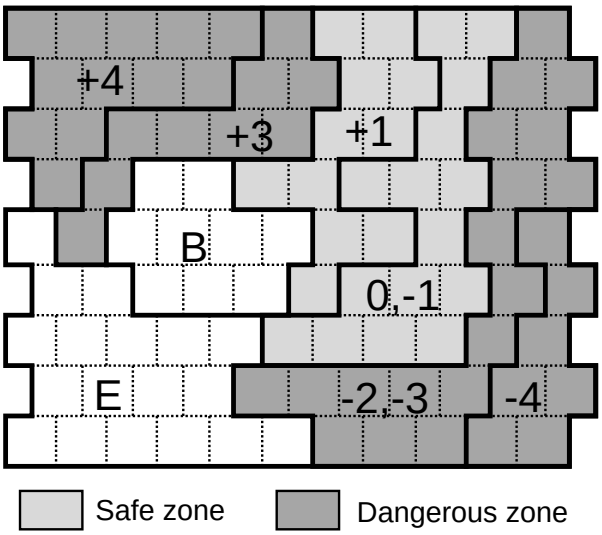


Figure 4.27: Map (11×9) partition for the insertion task in the tool pallet with positive and negative transversal error

for in two different ways: the resolution of the net may not be enough (a solution to this problem will be discussed in Sect. 4.4.4); but another explanation may be the physics of the problem itself and the fact that we are dealing with very small errors: for the cases with no offset and -1 mm, the correct insertion may not be exactly for 0 mm, but rather for some value in the range [0 mm, -1 mm]. In addition, due to a lack of symmetry of the tool, small negative errors will end up in a correct insertion, while the same errors in the positive direction will produce failure (the tool has a notch on one side that gets stuck in the claw of the clamping device, while it is rounded in the other side producing a compliant motion).

Errors for a new situation will be detected as explained for the previous case. The further the resulting activation pattern is from the central regions the greater the error (positive or negative) will be. For monitoring purposes, in Fig. 4.27 the different regions have been classified into safe or dangerous using grey levels.

4.4.3 Detecting states within insertion processes

In the previous section, error detection was based on computing the activation pattern for the whole sequence of states in an insertion task. Obviously, an error must be detected as soon as possible, while it is still possible to recover from it. For this purpose, the network can supply information about the present state in a process, since an activation pattern can be obtained

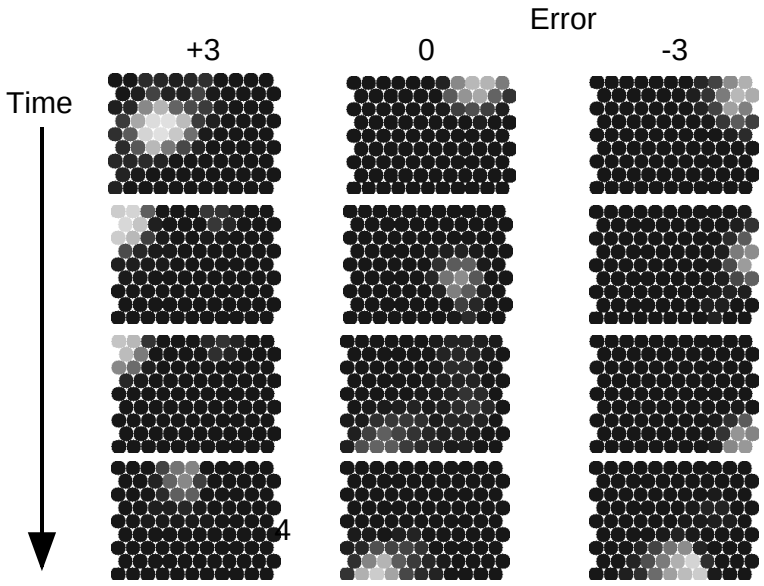


Figure 4.28: Temporal sequence of map activation for the insertion task in the tool pallet

for a particular instant. A sequence of these instantaneous patterns indicates the evolution of the process, and allows to detect failures as soon as they take place. This can be seen in Fig. 4.28: a motion picture for three cases is shown. The first column corresponds to offset +3 mm, the second to correct insertion, and the third to offset -3 mm. The temporal ordering for each state is indicated by the arrow. In the figure, it can be seen how for the first instant the upper and lower cases already have bright cells located in error zones, while this is even more evident for the second snapshot.

The evolution of the process can also be captured by the trajectories depicted onto the complete activation pattern. These trajectories have been shown in Figs. 4.24 and 4.26; they correspond to the winner cell for some instants. Using them, we can assign to every region on the map the moment at which it must activate. In this way the method is more reliable, since not only we check that the process remains within certain limits on the map, but also that it is constrained by certain temporal limits.

4.4.4 Improvements and generalizations

Increasing the resolution of the network

We faced above the problem that some different processes were not properly separated by the net, but instead they gave rise to overlapping regions. One

possible solution is to increase the resolution of the net by adding more cells. There are no published results about the influence of the number of cells on the behavior of a SOM network in a general case. The experience shows that increasing the number of cells will improve the resolution of the net, but obviously until a certain limit. This limit has to do with the nature of the problem: if two states are not separated in the input space, they will never be separated on the map (we have seen an example of this when dealing with the peg-in-hole task). However, the more neurons we use the greater the number of different error states the network will be able to identify. A state can be identified if at least one neuron has been labeled for that state.

The previous experiment with errors in the range [-4 mm, +4 mm] has been repeated for a larger network. The SOM has $22 \times 15 = 330$ units, hexagonally connected, with bubble neighborhood. The results are shown in Figs. 4.29 and 4.30. These results indicate that this map has a better resolution than the previous smaller one. The learning time has been kept reasonably small, using similar training parameters (ordering phase: $\eta = 0.2$ and $r_t = 15$; fine-tuning phase: $\eta = 0.04$ and $r_t = 5$).

The layout of clusters is similar to the previous case, with two regions for the beginning and ending states, and the rest of regions varying continuously with the error. The difference is that now the map can be easily partitioned into a number of regions corresponding to all the different error states. There is still some overlapping at the borders, that can be accounted for by the reasons mentioned in Sect. 4.4.2. The regions on the upper right corner may disappear with additional training, although in complex situations it might not always be possible to map the six-dimensional input space onto the two-dimensional output space in a perfectly continuous way.

Generalization to other types of errors

In all the previous experiments the network was able to recognize and identify all of the different types of errors that the network *had seen during training*. Two main questions arise: can the network detect error types that are not included in its training setΓ And, can the approach be generalized to other types of errors and combinations of these errors (e.g., along the OY and OZ axes)Γ

The answer to the first question is in the affirmative, provided that the signal pattern of the new error is different from any other signal pattern, since the network cannot distinguish states with very similar input patterns. Then when a different new pattern s fed into the network, no neuron vector

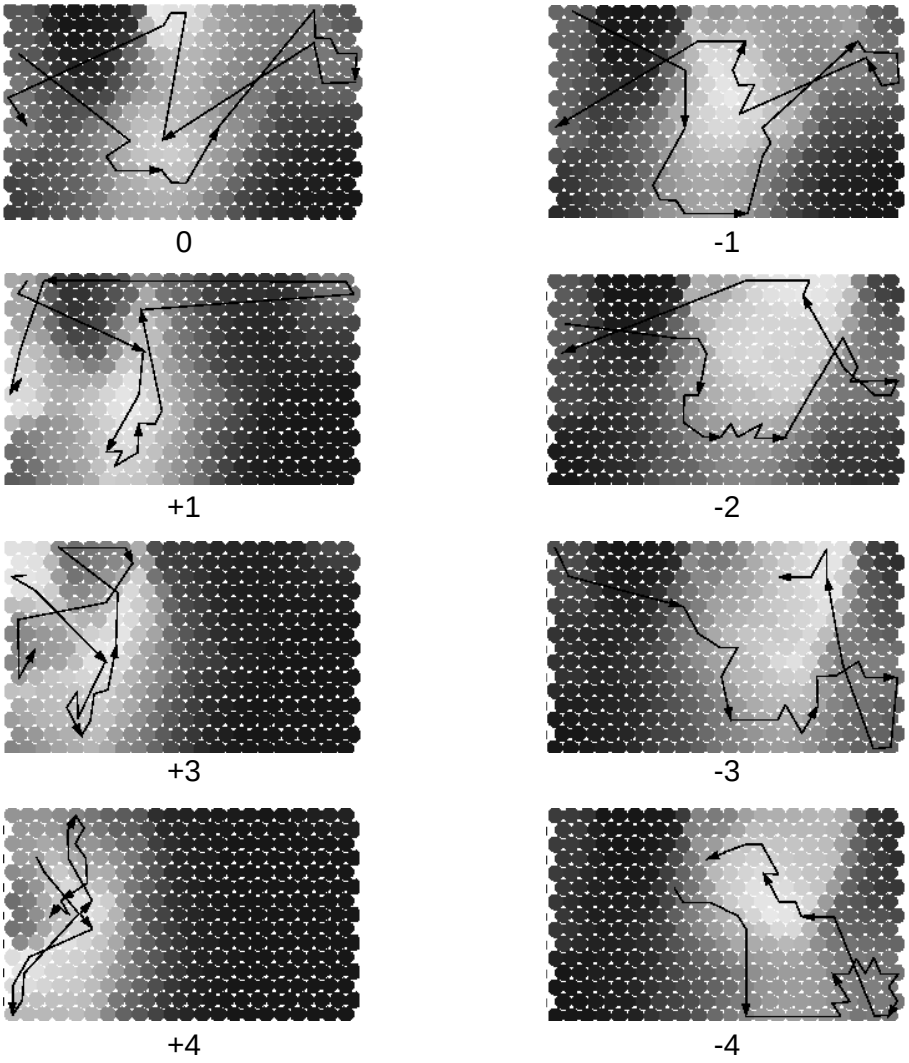


Figure 4.29: Map activations and traces of winner units for the insertion task in the tool pallet with positive and negative transversal error

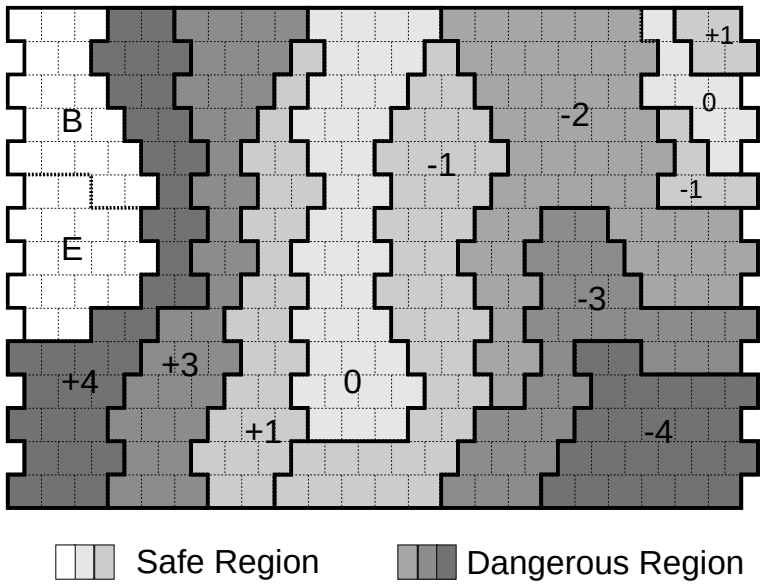


Figure 4.30: Map (22×15) partition for the insertion task in the tool pallet with positive and negative transversal error

will be close enough to the input vector, i.e.,

$$||x - w_c|| = \min_i\{||x - w_i||\} > \mu \tag{4.3}$$

where μ is a fixed application-dependent threshold. This means that the network does not know what is happening, but it knows that something is going wrong. At least the network detects that it does not know the new state and this is a very important property, since the state will not be misclassified. Instead, an *unknown state* warning (i.e., *threshold overflow*) will be notified. This warning suffices for detecting any unseen type of error condition.

As regards generalization, it is possible in principle to learn to identify as many states as the number of neurons in the network. The main constraint is that the input pattern for a certain error state must differ from those of all the other states; otherwise, the same neuron will respond to different states.

A combination of errors along OY and OZ axes does not affect the success of the approach. The only effect of the OZ error component is that the time instant in which contact between tool and pallet takes place is delayed or advanced. After contact, the situation is exactly the same. This is easily controllable by monitoring when the winner neuron gets out of region B (guarded move). This situation will be general as long as the OZ axis is made to correspond to the direction along which the gripper approaches its destination and on which the distance between them is measured.

An error along the OX axis is not relevant to our case in the flexible manufacturing system, since the main source of error is the movement of the vehicle that carries the tool-pallet. Obviously, this motion does not affect the horizontal plane on which the pallet is located. In a general case, an error along OX is not a problem since this type of error will give a very different input pattern from those caused by other errors.

The network is also able to generalize and cope with arbitrary combinations of these errors. Let us suppose that a particular error along the OY axis is detected by a certain neuron error along the OX axis is detected by a certain neuron N_2 on another location on the lattice. An error along both the OX and OY axes will correspond to a point in sensor space located in a region somewhere in between those corresponding to an error along OX and along OY. The SOM theory demonstrates that — after training — neurons which are close on the map respond to inputs which are close on the input space. Then, the neurons which, after training, are positioned between N_1 and N_2 are responsive to the combined action of the inputs for OX and OY.

To achieve an adequate identification for combinations of errors, samples for these combined errors must be used in the training phase. Therefore, the training set includes errors along OX and OY, as well as combinations of both; that is, sample errors taken from a grid in plane XY. A uniform distribution of these samples on the plane guarantees that the SOM covers all types of combined errors.

Since there is no limit in the number of neurons used to increase resolution, the error distinction is only limited by the noise of the sensor signals. Since the topology of the sensor space is preserved, the response of the map changes smoothly from neuron N_1 (for an error along the OY axis) to the intermediate neurons, as this error decreases and error along the OX axis increases, until the maximum response is given by N_2 when the error is caused only along the OX axis. The topological order is preserved as long as the high-dimensional input space can be mapped onto the two-dimensional lattice. This is not always possible, e.g. a spherical three-dimensional region cannot be mapped continuously on a rectangular lattice. In this case, all of the regions are represented as separate parts, but there are neighborhood relationships which cannot be reflected by the map (discontinuities); in any case, different neurons will distinguish the different error states and any combination of errors will be identified too.

Real-time aspects

The low computational complexity of this approach makes it feasible for its use in a real-time environment. The training phase is the more computer-

intensive process, which can be done off-line. Afterwards, the monitoring phase only requires the comparison of the input vector with the weight vector of all the neurons of the network in order to choose the winner cell, i.e. the state of the input pattern. Since the number of neurons is unlikely to be more than a few hundreds, this operation can be done on a personal computer in a few milliseconds.

Instead of training the network off-line with a previously collected set of input samples, the training process can be done on-line. It is feasible to train the network at the same time as the input signals are obtained from the sensors. After a learning phase of a few minutes, the network is ready to monitor the process without further training, as long as new states do not appear. If an unknown state is detected, the training process might be re-run to learn this new state on-line. Thus, new error states are dealt with as soon as they appear. Minimal feedback with the operator is required in order to provide a label for the new state. A more complete system could send the information from the neural network into a high-level reasoning system to perform more complex operations such as planning and prediction.

4.4.5 Discussion

A novel approach for error detection in plan monitoring based on SOMs has been presented in this section. Input data are provided by the six force and torque signals from the wrist sensor of the robot arm. The method has proved to work properly in complex real-world situations involving fine motion and grasping, for which a geometric analytical model may not be feasible, or too difficult. It is been used for an actual assembly task like a complex insertion operations in a flexible manufacturing system. The influence of the size of the network on its resolution has been also discussed.

The training of the network is very straightforward, because there is no teacher. We have shown how the network must be labelled in order to be used to monitor the process. Despite the simplicity of the network, the self-organizing map manages to store complex information about the states of the task, and allows to detect error conditions easily.

Due to the short time required for the learning process and its simplicity, it allows for a great flexibility and new tasks can be readily learned without a long process of analysis. In order to achieve a good training we would need examples of every error situation. If this is not possible we still can detect abnormal samples which differ from those learned by the network. In that cases, no neuron will respond to the signals, so the network will be idle. We can associate this situation to an abnormal functioning and stop the task. Then, we could train the network with the last signals presented, so the

network will learn this new situation.

It must be noted that the network does not store any temporal information. However, because of its topological properties, the signal sequences lead to smooth trajectories on the map. Occasionally, there are jumps in the map, due to sudden signal changes, or topological restrictions, as a 6-dimensional subspace is being mapped onto a 2-dimensional surface. Thus, the process can be monitored by observing this trajectory between clusters.

An interesting extension would be modifying the training procedure to allow the network to learn on-line when the task is running, and do the labeling process automatically. We are also dealing with temporal sequences of signals, so we could use the temporal information associated to the signals, e.g., by adding more inputs with the values of the signals in past instants.

Error detection is a previous and necessary condition for error recovery. An extension to the system will be the addition of recovery actions associated to cells of the network labelled with error states. A simple synthetic example is introduced in the next section, but a real, autonomous architecture will be presented in Chap. 5.

4.5 Building a SOM-based fine motion plan

This part addresses the problem of associating actions to the different states of a fine motion task. A qualitative method is used to show that, in the absence of precise quantitative information, a controller is able to perform an insertion task like the peg-in-hole problem [Cervera and del Pobil, 1995a,b]. In order to cope with uncertainty, however, the introduction of a perception-based approach is required, and the SOM provides a method for integrating perceptual information and qualitative knowledge.

This approach is illustrated with a simple simulation model, introduced in Sect. 4.5.1. Initially, attention is paid to geometry, but the final goal is a perception-based plan which can cope with geometric uncertainty.

Section 4.5.2 describes the purely geometric approach, and a qualitative spatial reasoning method to control the insertion task. However, it is shown that, in the presence of position and control uncertainty, geometric information is not sufficient to guarantee a successful plan.

In Sect. 4.5.5, a perception-based approach is presented. The use of force sensing provides the information required to properly identify the state of the system, thus solving the uncertainty problem. A neural network-based system is used to map the raw sensory data to the state space. As a final result, a robust qualitative approach to reasoning about manipulation involving contact is derived. It is based on perception, without almost any

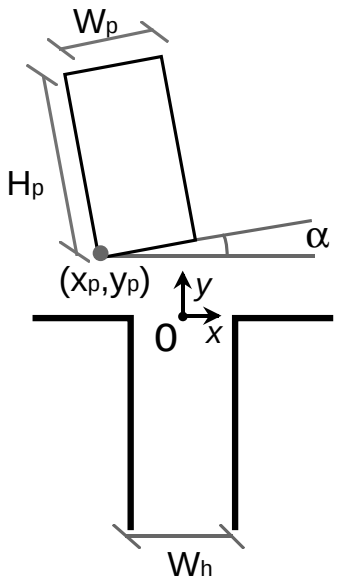


Figure 4.31: Geometry of the 2D peg-in-hole insertion task

geometric information, thus being adequate for implementing in a real robot in a three-dimensional environment, where the geometry is more complex.

4.5.1 Qualitative model of a manipulation task

In this section a simulated insertion task is shown as an example of manipulation. With simple geometric knowledge, we show how to build a qualitative spatial reasoning system to perform the insertion task. However, as this model relies in positional measurements to calculate the state of the system, high accuracy is required, which usually is not available in real systems.

In Fig. 4.31 the geometry of the model is shown. It is a rectangular peg of width W_p and height H_p, to be inserted in a chamferless hole of width W_h and infinite depth in an xy plane. Let (O, xy) be a coordinate system attached to the hole. The coordinates of the peg (x_p, y_p) are given by its left-down corner, and the orientation by the angle α with X axis. In our simulations we only consider positive angles in the interval $]0, \pi/2[$, since negative angles only add symmetry to the problem. Thus, the peg is described with coordinates (x_p, y_p, α), which define the configuration space.

A constraint equation set is defined by the physics of the system. Each equation describes a different contact between peg and hole. Figure 4.32 depicts these constraints and three examples of the configuration space for different peg angles. The shadowed zone is, together with the Y-positive

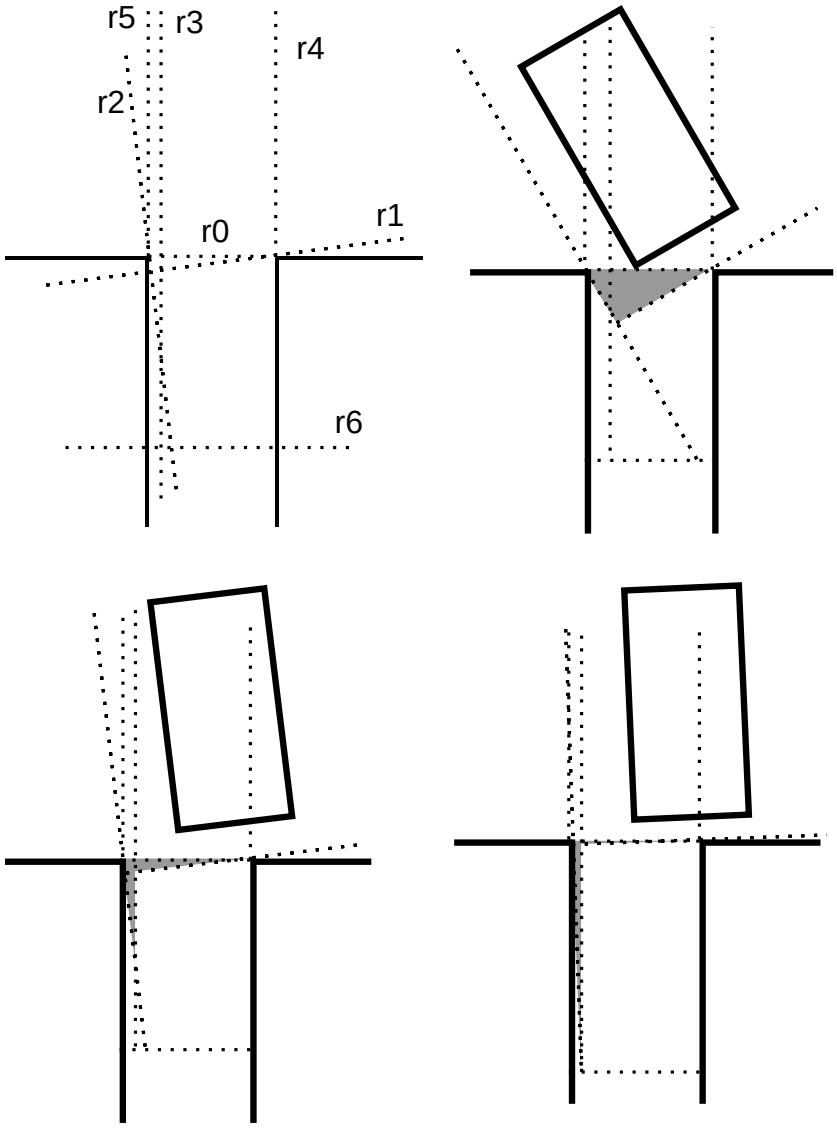


Figure 4.32: Graphical representation of the constraint equations. Three different configurations of the peg, with the allowed regions inside the hole

semiplane, the allowed peg motion space. The contact constraint equations are the following:

- Left-down peg corner and horizontal hole surface ($r0$):
$$y = 0$$
- Bottom peg side and right hole corner ($r1$):
$$-x\tan(\alpha) + y + \frac{W_h \tan(\alpha)}{2} = 0$$
- Left peg side and left hole corner ($r2$):
$$x + y\tan(\alpha) + \frac{W_h}{2} = 0$$
- Right-down peg corner and vertical hole surface ($r3$):
$$-x + \frac{W_h}{2} - W_p\cos(\alpha) = 0$$

Two more constraints are added to specify the hole interval:

- Left hole side ($r4$):
$$x - \frac{W_h}{2} = 0$$
- Right hole side ($r5$):
$$x + \frac{W_h}{2} = 0$$

And finally another constraint describes the goal of the system. The peg is completely inserted when its left-up corner is in the hole:

- Left-up peg corner in hole ($r6$):
$$y + H_p\cos(\frac{W_h - W_p}{H_p}) = 0$$

These constraints define a partition of the configuration space, as shown in Fig. 4.33. The elements are labeled from $S0$ to $S6$, and D is the goal region (these labels should not be confused with the contact states p1-6 of the previous experiments). The regions are defined uniquely by the following expressions:

$$\begin{array}{rlllllllll}
\text{S0:} & (r0 \geq 0) & \text{and} & (r5 < 0) & & & & & & \\
\text{S1:} & (r0 \geq 0) & \text{and} & (r5 \geq 0) & \text{and} & (r3 \geq 0) & & & & \\
\text{S2:} & (r0 \geq 0) & \text{and} & (r3 < 0) & \text{and} & (r4 < 0) & & & & \\
\text{S3:} & (r0 \geq 0) & \text{and} & (r4 \geq 0) & & & & & & \\
\text{S4:} & (r0 < 0) & \text{and} & (r1 \geq 0) & \text{and} & (r2 \geq 0) & \text{and} & (r3 \geq 0) & & \\
\text{S5:} & (r0 < 0) & \text{and} & (r1 \geq 0) & \text{and} & (r2 \geq 0) & \text{and} & (r3 < 0) & & \\
\text{S6:} & (r1 < 0) & \text{and} & (r2 \geq 0) & \text{and} & (r3 \geq 0) & \text{and} & (r6 \geq 0) & & \\
\text{D:} & (r6 < 0) & & & & & & & &
\end{array}$$

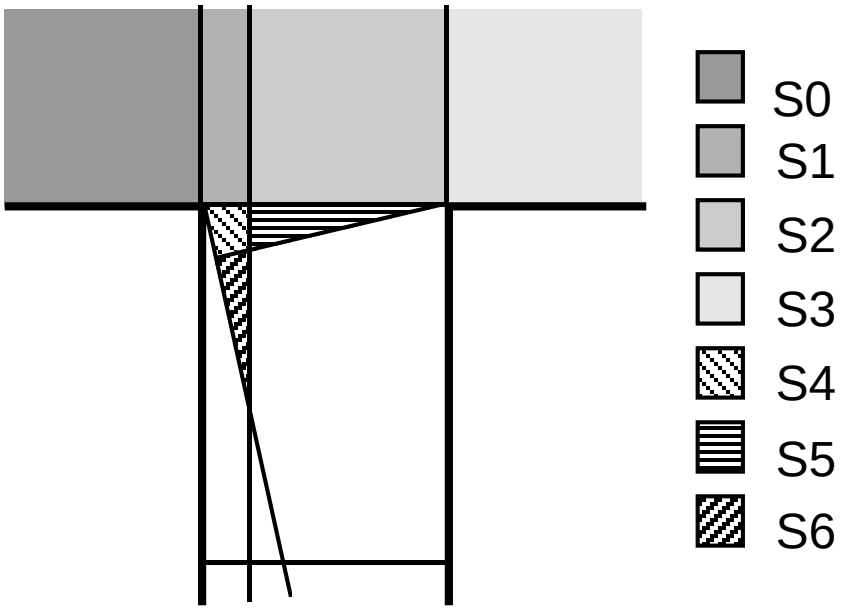


Figure 4.33: Configuration state partition for a sample angle

Now, if the coordinates and orientation of the peg are known, we can calculate the plane region where the peg is. This region is limited by several constraint equations. Each constraint defines an edge between two regions, thus a motion between regions can be detected by the value of that constraint. This knowledge is used in the next section to build the qualitative model of the system.

4.5.2 Qualitative model

The constraint set and partition which was previously defined allows us to build a qualitative model of our system, a graph of states and transitions.

States

There are two classes of states, *free* and *contact* states. Free states are characterized only by the plane partition where the peg is, thus all states from [S0] to [S6] are free states. When the system is in a free state it has 3 Degrees of Freedom (DOF) and every possible motion is allowed, a motion being a continuous change of variables (x_p, y_p, α).

Contact states are those states where one or several contacts between the peg and the hole exist. These states are characterized by the tuple consisting on the region of the partition and the appropriate constraints which define

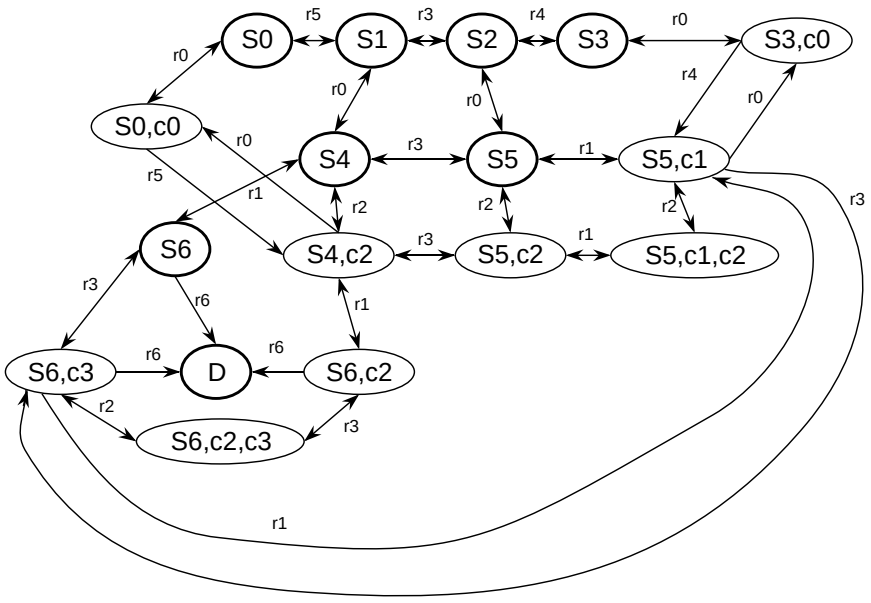


Figure 4.34: State space diagram

the contacts (e.g. [S0,c0] or [S6,c2,c3]). Of course, the constraint contact equation must be zero-valued. Thus, state [S0,c0] is defined by constraint ($r0 = 0$), whereas state [S0] was defined by ($r0 \geq 0$). From a contact state, only those motions which will lead to a zero-or- positive value of the contact constraint are allowed, remaining in the same state or moving to another. Each contact subtracts one DOF from the system.

Thus a compliant motion is possible if there are only one or two contacts between peg and hole. However, for the sake of simplicity, we only allow translations or rotations, but not simultaneously. For example, in state [S5,c1,c2], a simultaneous translation and rotation would be possible while keeping both contacts, since the system has 1 DOF. However, this is not allowed: the peg cannot be rotated, rather it can only be translated, thus breaking one of the contacts, or both.

Transitions

A transition occurs when a constraint equation value becomes zero or when it is zero and is increased. States and allowed transitions are depicted in Fig. 4.34. Possible initial states are [S0], [S1], [S2] and [S3]. The destination state is labeled with [D].

In this state space, several transition types are allowed depending on whether the involved states are free or contact states. It is assumed that

only one constraint equation changes at a time. This is not true in the case when there are two contacts and the peg translates breaking them both, e.g. from state [S5,c1,c2] to state [S5], but we can think that the system goes through [S5,c1] or [S5,c2] in a zero time.

Another case when several constraint equations change simultaneously when a *singularity* takes place. This occurs in compliant motions when a peg corner reaches a hole corner. In this case there is a sudden change of motion direction, and we assume that the compliant motion goes on. For example, the peg is at the left of the hole, in state [S0,c0], and is moved to the right in contact with the surface. When the peg corner in contact with the surface reaches the left hole corner, there is a sudden change and the peg falls in the hole. But if we assume that there is a compliant motion, the left peg side keeps in contact with the left hole corner. The system has changed from state [S0,c0] to state [S4,c2].

Singularities are depicted in our diagram with two separate arrows, because two different constraint equations are involved, depending on the direction of motion. Every other transition only has one constraint equation, and the change of sign indicates the direction of the transition.

Finally, between two (topologically-)neighbor free states, the transition is given by the constraint equation common to both states; and between a free state and a contact state, it is given by the appropriate contact constraint equation. Now, our method of qualitative spatial reasoning is nearly completed. All we need is to find a path in the state space graph from an initial state to the destination state. This path will consist of a sequence of states and transitions, each transition being a constraint equation. If we knew the motions which cause every equation to change, we could issue these motions as commands to the arm controller, and the system will move following the path to the destination.

Qualitative motions

Given all the constraint equations, we can calculate how they change when the coordinates of the system change, by calculating the partial derivatives with respect to (x, y, α). In Table 4.13, the signs of these derivatives are shown. Only this information is needed in a qualitative system. We cannot specify the magnitude of the change but only its sign, i.e., positive, zero, or negative.

Now, if we want to change, for example, from state [S5] to state [S5,c1], we first see that the constraint equation involved is $r1$, and it should be decreased, because [S5] is defined by $(r1 \geq 0)$ whereas [S5,c1] is defined by $(r1 = 0)$. If we look at Table 4.13, we can see the derivatives of $r1$ with

Table 4.13: Qualitative values of constraint equation derivatives

	$\partial r0$	$\partial r1$	$\partial r2$	$\partial r3$	$\partial r4$	$\partial r5$	$\partial r6$
∂x	0	−1	+1	−1	+1	+1	0
∂y	+1	+1	+1	0	0	0	−1
$\partial \alpha$	0	+1	−1	+1	0	0	0

respect to (x, y, α). Then, in order to decrease $r1$, we should decrease x, increase y or increase α. Thus, given a pair of states and the appropriate constraint equation, we can calculate the changes in the variables to fire that constraint equation (change its sign) and move to the destination state.

A problem in a qualitative approach is that we cannot control the amount of change, that is, the motion step. Thus, a big step could jump over several states. For example, a moderate negative change in x could easily lead from state [S2] to state [S0], because [S1] is quite narrow. Our strategies should be robust with respect to these possible jumps.

Another more important problem is that a variable change affects not only that constraint equation but others as well. This can also be seen in the table. A change in x produces changes in r1, r2, r3, r4 and r5. This is important, for instance, if we are in state [S4,c2] and want to move in compliant motion to state [S6,c2], then we should fire constraint r1, but not r3. In this case, however, both x and y change, and r3 is affected, and could fire before r1, thus leading to an undesired state [S5,c2]. We should follow a strategy that minimizes those undesired transitions. One general principle is to use compliant motion, because, as the system has less DOFs, the possible number of transitions is reduced, thus providing a more reliable path. With this strategy, we can build the motion plan shown in Fig. 4.35.

In this plan, each node consists of a state and an action. The states are those belonging to the state space, and the actions are qualitative changes of the variables, with or without a constraint equation if it is a compliant motion. This plan provides the adequate qualitative motion commands to move the peg from any initial position and orientation to its destination, with only one added condition:

$$BackStep < \sqrt{W_h^2 - W_p^2} \tag{4.4}$$

BackStep is the amount of motion when going backward from a two-contact state (namely [S5,c1,c2] and [S6,c2,c3]), and it guarantees that the peg will not go completely out of the hole.

The step size used in the other states is not limited, but it is assumed that the force control stops the arm motion when contacting the peg and the

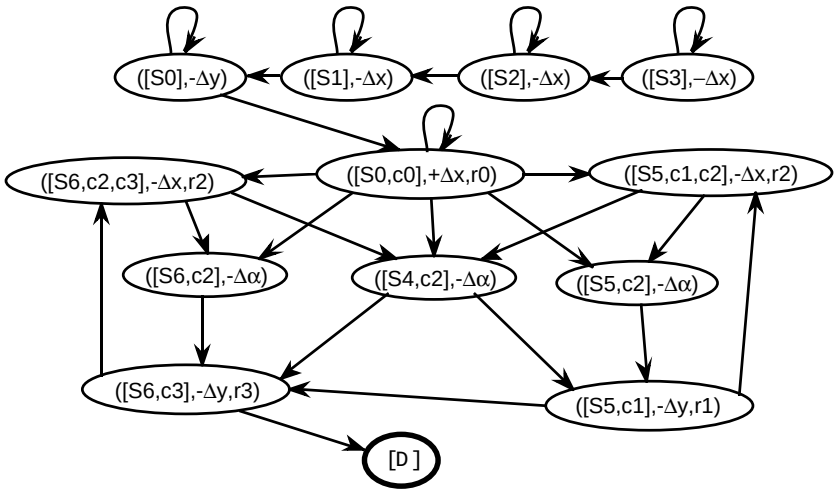


Figure 4.35: Qualitative motion plan based on position

hole. Thus the system can go, in one step, to a state not connected to the initial one in diagram of Fig. 4.34 (e.g. from [S0,c0] to [S5,c1,c2] through [S4,c2] and [S5,c2]). Our plan takes this into account and includes transitions to all the states where the system might evolve after a single step.

It is also assumed that the rotations are limited by the contacts. For example, when moving from node $([S4, c2], -\Delta\alpha)$ to node $([S6, c3], -\Delta y, r3)$, the system evolves through states [S4] and [S6], but stops when contacting with $r3$. Translation motions can be free or compliant. In compliant translations, only one variable change is given, but the other one's change (if needed) can be calculated from the equations. Nevertheless we assume that the arm force control provides the compliant motion with only one indication.

4.5.3 Simulations without uncertainty

This qualitative system has been tested in a number of simulations with different parameters, and has been proved to work properly. One of such simulations can be seen in Fig. 4.36, which depicts a sequence of the peg moving until it is inserted. The state sequence in this simulation is:

$$[S0] \rightarrow [S0, c0] \rightarrow \underbrace{[S5, c1, c2] \rightarrow [S5, c2] \rightarrow [S5, c1]}_{6 \text{ times}} \rightarrow$$
$$\rightarrow [S6, c3] \rightarrow [S6, c2, c3] \rightarrow [S6, c2] \rightarrow [S6, c3] \rightarrow [D]$$

As the initial angle is big, the peg bottom contacts with the right hole corner several times. Each time it contacts, it moves backwards and rotates. Thus the angle is reduced until the peg can enter into the hole and contacts

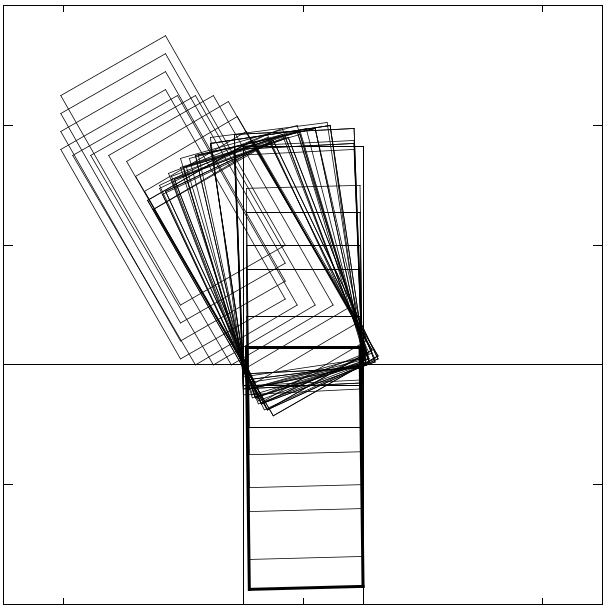


Figure 4.36: Simulation of peg insertion without uncertainty, guided by the qualitative plan

with the right inner surface. Then it moves backwards again and rotates, and finally it enters completely.

4.5.4 Uncertainty and perception

It has already been mentioned that the abstract peg-in-hole problem can be solved quite easily if the exact location of the hole is known and if the manipulator can precisely control the position and orientation of the peg. We have shown how it is solved under a qualitative perspective. But even with this approach we are relying on precise values of the coordinates of the system, which are necessary to calculate the constraint equations and obtain the system state. Small errors in these values can lead to calculate incorrect states and incorrect actions, thus making all our strategy become useless.

Perceptual information would allow us to reduce the geometric uncertainty and to determine the state of the system with greater reliability. We could also monitor the actions and detect the error conditions, e.g. undesired contacts. In order to detect contacts, we should measure the forces between the peg and hole. Wrist-mounted force sensors are usually used to measure the external forces and torque applied to the end effector as it interacts with the environment, so they are very suitable for our purpose.

We simulate a force/torque sensor attached to the top of the peg, as

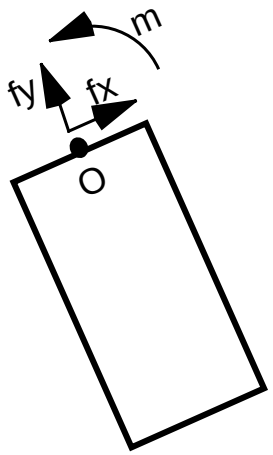


Figure 4.37: Simulated force sensor

Table 4.14: Correspondence between contact states and positional states

P0:[S0], [S1], [S2], [S3], [S4], [S5], [S6]
P1:[S5, c1]
P2:[S4, c2], [S5, c2], [S6, c2]
P3:[S5, c1, c2]
P4:[S0, c0], [S3, c0]
P5:[S6, c3]
P6:[S6, c2, c3]

depicted in Fig. 4.37. This sensor gives us the measures of force (f_x, f_y) along two axes of a coordinate system attached to it and a torque signal m with respect to O. We are interested in identifying contact states with the help of the force sensor. These contact states were introduced in Fig. 4.2 on page 65. The no-contact state (only the weight force) is considered too. We assume that no friction forces exist, nor inertial effects by the peg motion.

If we could identify the contact state, we could almost calculate the positional state in our qualitative model with no need for any other information. There are some cases when the contact state gives us the positional state, but this is not always possible because a contact state may correspond to several different positional states. In Table 4.14 this correspondence is shown.

Obviously the sensor does not provide us directly with the contact state, rather it gives us three raw signals of force and torque. An independent problem is to map these signals to the contact state appropriately, but experiments in previous sections have shown the feasibility of using SOMs for this purpose.

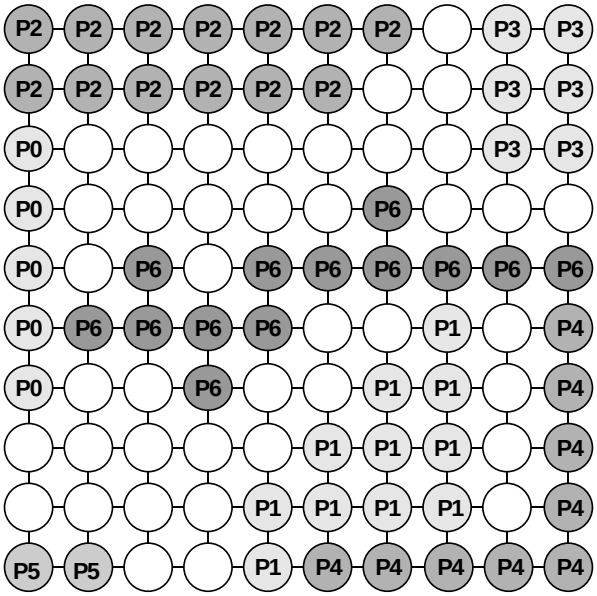


Figure 4.38: Trained and labeled neural network

Although we are dealing with a simulated problem, our aim is to implement our model in a real system. This implementation is carried out in Chap. 5. The main differences are that in a general case the contact states are unknown (the network cannot be labeled) and the plan is not known a-priori, but it must be learnt through self-experience.

In these experiments, a 10×10 network has been used, with three different data sets for training, labeling and testing of 2,100 samples each, randomly selected from all the contact states. Sensor measurements f_x, f_y and m were inputs. Distance function was the Euclidean norm. Forces were calculated for angles from 0 to 45 degrees and reaction forces between 0 and 5 kgf. Peg weight was 2 kgf. Since network inputs are absolute measurements, if these parameters change, the system should be retrained.

Figure 4.38 depicts the structure of a trained and labeled network. Labels were assigned according to the class with the greatest number of wins. Each contact state forms a cluster. Classification results are given in Table 4.15 (the performance is excellent since friction is not considered).

4.5.5 Perception-based qualitative model

Although mapping directly from contact states to positional states is not possible, we get all the necessary information for our plan, as all its states

Table 4.15: Classification results of testing set with a 10×10 network

	P0	P1	P2	P3	P4	P5	P6
P0	100	–	–	–	–	–	–
P1	–	90.7	–	–	6.7	–	–
P2	–	–	100	–	–	–	–
P3	–	–	–	100	–	–	–
P4	–	5.3	–	–	93.3	–	–
P5	–	–	–	–	–	100	–
P6	–	4.0	–	–	–	–	100

can be properly classified with the exception of two subsets:

$$\{[S0], [S1], [S2], [S3]\} \quad \text{and} \quad \{[S4,c2], [S5,c2], [S6,c2]\}.$$

To solve the first ambiguity we need an uppermost bound of the geometric uncertainty, which usually is easy to provide. If such a limit exists, the only thing to do is to move the peg to the left, at the beginning of the plan, for a distance greater than this uncertainty limit, in order to ensure that the peg will stay at the left of the hole. The second ambiguity is even simpler to solve. As shown in Fig. 4.35, states [S4,c2], [S5,c2] and [S6,c2] require the same action ($-\Delta\alpha$). We can join these three nodes in a unique one with the same action.

Finally, state [D] should be identified by geometric information with an uncertainty bound, which is reasonable, since uncertainty is supposed to be much lower than peg height. We are working on an extension to our model which includes finite hole depth, thus allowing to detect this goal state by the contact with the bottom of the hole.

The resulting perception-based qualitative plan is shown in Fig. 4.39. Each state is now identifiable with only the measurements of the force sensor, but this does not exclude the use of the geometric information. One might combine these two methods in obtaining the state to increase system reliability.

This new plan still requires the assumption about the backstep maximum length, which could be a critical requirement if the clearance is very small, since the maximum step might be lower than the precision of the control. Further work is necessary to study other planning strategies which will avoid this requirement.

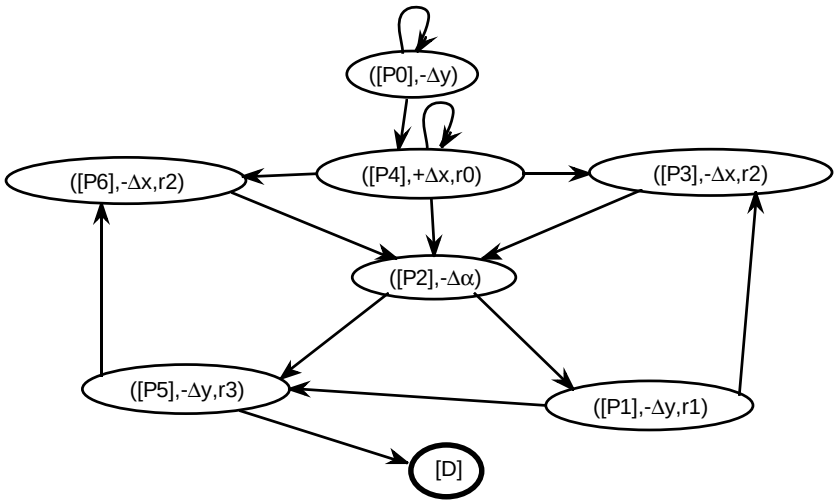


Figure 4.39: Perception-based qualitative motion plan

4.5.6 Perception-based simulations

Several simulations have been performed like those in Sect. 4.5.3, but now the state information is obtained from the sensory data. Figure 4.40 depicts the sensory data used in a simulation, where the state sequence shown in that section can be properly identified from the force and torque signals. During these simulations, all the states were correctly identified.

The controller behavior is equivalent to that of the geometric one. The actions are the same but now there is no need for information about the position and orientation of the peg. The peg, however, must be located at the left side of the hole at the beginning, the backstep cannot exceed the limit stated before, and the destination state should be detected by geometric measurements.

4.5.7 Discussion

This section has presented a qualitative reasoning approach to motion in contact based on simple geometric relationships. It has been applied to solve an insertion task, common in robotic manipulations. To deal with uncertainties in position and control, a perception-based qualitative control method for manipulation tasks with uncertainty has been introduced. This method has been proved to be robust and it solves the uncertainty problem that arises when using only geometric information. Sensory data is useful for describing the state of the system, and is less sensitive to errors than position information. A SOM is used to translate physical measurements to symbolic

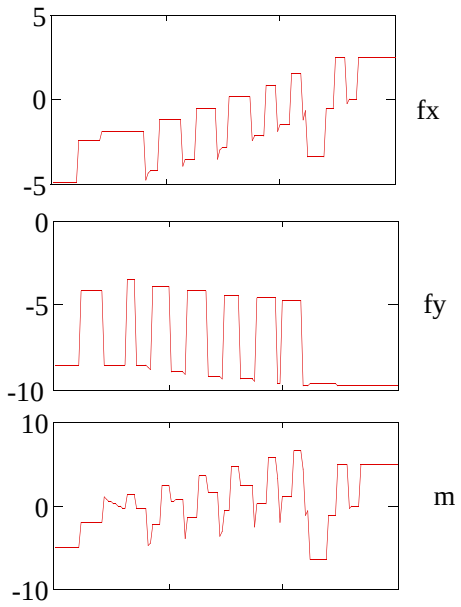


Figure 4.40: Force measurements in a sensor-based simulation

information.

In real-world applications it is necessary that the plan is learnt by the robot in an autonomous way. This approach is opposed to the pre-computing of a plan and the use of sensing only for corrections at run-time. If the plan is learnt, states are identified from real-world situations, thus the plan copes with any uncertainty or unmodeled features. This is the purpose of the architecture which will be introduced in the next chapter.

Chapter 5

Robot learning architecture

The simulations presented in the previous chapter were restricted to two dimensions for simplicity. Despite this fact, they were illustrative of the capabilities of the SOM for the extraction of features suitable for identification of contact states. Though the set of contact states was known in advance, the SOM learnt the features in an unsupervised manner. This is very important for more sophisticated problems (in three dimensions) since the number of states increases considerably and becomes difficult to manage explicitly.

In real-world situations, a pre-builded plan like the one described in Sect. 4.5 is unlikely to be effective. The robot can extract some features from the sensor signals, and it needs to learn the plan by itself too. There is no model of the world, but the robot builds an internal representation with the SOM and the policy that it learns. The robot must learn the actions which, according to the state extracted from the features, will attain the goal in the best way defined by some criteria (reinforcement). This is a big improvement over the monitoring scheme presented in the flexible manufacturing system.

5.1 Situated, embodied agents

Well beyond the ethereal world of computer algorithms, a robot is what we call a *situated, embodied agent*: a physical, material system placed in a specific real environment: it performs real operations, and it has to manage to the best of its ability. This agent has two basic capabilities: it can observe the world through its sensors, and it can act based on these observations in order to achieve a particular goal. Actuators and sensors are the bridges, one for each direction, between the world and the agent (see Fig. 5.1). They first limit what the agent is able to know and to do. Only information acquired through sensing can increase the agent's knowledge which, at the beginning,

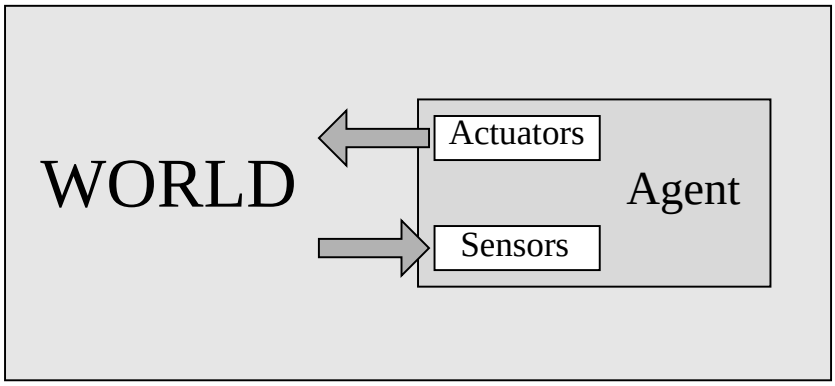


Figure 5.1: Diagram of a situated-embodied agent

consists of programmed data and algorithms. And, obviously, an agent can do only those actions allowed by its actuators (the most sophisticated driving control won't make an agent move if it does not possess motors and wheels!).

Theoretically, an agent could perform a task without sensing devices, if it is programmed in advance with the correct sequence of actions. But in real-world tasks the presence of uncertainty makes such hard-coded programs fail miserably. Uncertainty occurs when an object is located a few millimeters away from where it should be according to the agent's internal model, when the agent moves to the right more than it was commanded, or when noise disturbs a sensor signal with a value which does not correspond with the actual observable magnitude. Uncertainty can be diminished with careful and accurate design, but it is practically impossible, or just too costly, to be completely eliminated.

The agent's behavior is not limited to the execution of a given plan, but it should profit from its capacity to observe the world in order to perform the best action according to its current goal(s). Thus, during all its lifetime, the agent executes perception-action cycles: first, it observes its environment; then perception is evaluated according to an objective and, finally, an action is performed which, to the agent's knowledge, is a step towards the achievement of the goal.

For an external observer, the agent essentially performs a mapping function between sensations and actions. If the agent's structure is intended to be more complex than a black box, some kind of abstraction is needed. Sensors and actuators are provided *as is*, they cannot be changed at all; as a consequence, it seems obvious that both the processing of sensor signals and the generation of control signals are tightly coupled with the nature of the devices.

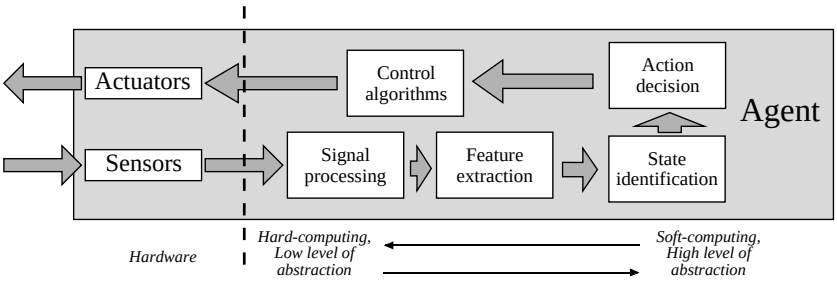


Figure 5.2: Levels of processing in the agent

Aside from the case of simple reactive control, when sensing and actions are tightly coupled, action decision is an abstract process which depends on some features of the current sensing signals and a specific internal state of the agent. This is a complex process since it must take into account the efficiency of the agent's behavior; it is desirable that agents improve its performance by learning from its previous experience and by exploring new actions. A possible representation of this levels of abstraction is depicted in Fig. 5.2. There is only one sharp limit between hardware (sensors and actuators) and software. The software levels are not intended to define a strict hierarchy, but a design principle.

Low levels are dependent on the particular problem and agent architecture; they are programmed according to the available task knowledge and engineering guidelines to perform their particular job, with local criteria of performance. The signal processing module should extract the maximum information it can from the available sensors, while rejecting noise. The control algorithms should issue the appropriate signals to the actuators to perform all the possible motions, to the highest requirements of accuracy and speed.

Feature extraction is the selection of the original signals into more effective features for a particular purpose, which, in this case, is the achievement of the goal by the agent. However, except for very simple problems, there are no proper criteria for evaluating the effectiveness of features and generate systematically the extraction process. Thus the selection of features becomes to a great extent problem-oriented.

High levels are intended to be general and thus applicable to different tasks and agents. Information at this level is made of features, states and actions, which are loosely, or not at all, coupled with the physical signals. Initially, the agent is programmed with a basic behavior (a mapping from states to actions) which is improved through experience. This modules evaluate the history of states, actions and the performance of the agent in achieving the task, and basically the state-action mapping is modified with the intention

of improving future executions of the task.

The solution of the task cannot be programmed due to uncertainty and/or lack of knowledge. The agent should discover an efficient solution despite the uncertainty. It carries out actions in an autonomous way, but it does not get an explicit feedback of the error, nor does it know which the correct solution is. Only a measure of the evaluation of the task is provided to the agent (this is called learning with a critic as opposed to learning with a teacher, when it is provided with the correct solution).

The agent has to be provided initially with:

- an exploration strategy
- a learning mechanism
- an evaluation of its performance

The final goal is to build an autonomous agent which is capable of learning from its own experience. It must be kept in mind that a significant amount of task knowledge is embedded in the architecture: the selection of features and control actions, the exploration, learning, and evaluation methodologies. At the beginning the agent's skills are limited, and random exploratory actions must be carried out frequently. After several trials, either successful or not, the agent should be able to learn a policy (correspondence between states and actions) which tends to improve its proficiency.

5.2 An architecture for a manipulator robot

The application of the previously stated principles to the particular domain of robotic manipulation poses the necessity of specifying concrete methods and algorithms for each of the modules. Starting from the specifications of the task and the physical robot (the agent), its available sensors and actuators, the control modules and the signal processing algorithms should be designed to maximize the use of the physical system for the desired tasks. High-level modules should exhibit the following characteristics: simplicity, robustness, efficiency, and generality. The adaptation to a particular problem is performed by the lower levels of processing. Based on the previous characteristics, and the principles of the general design, a diagram of the specific architecture for the manipulator robot is depicted in Fig. 5.3. Each module is described thoroughly in the following.

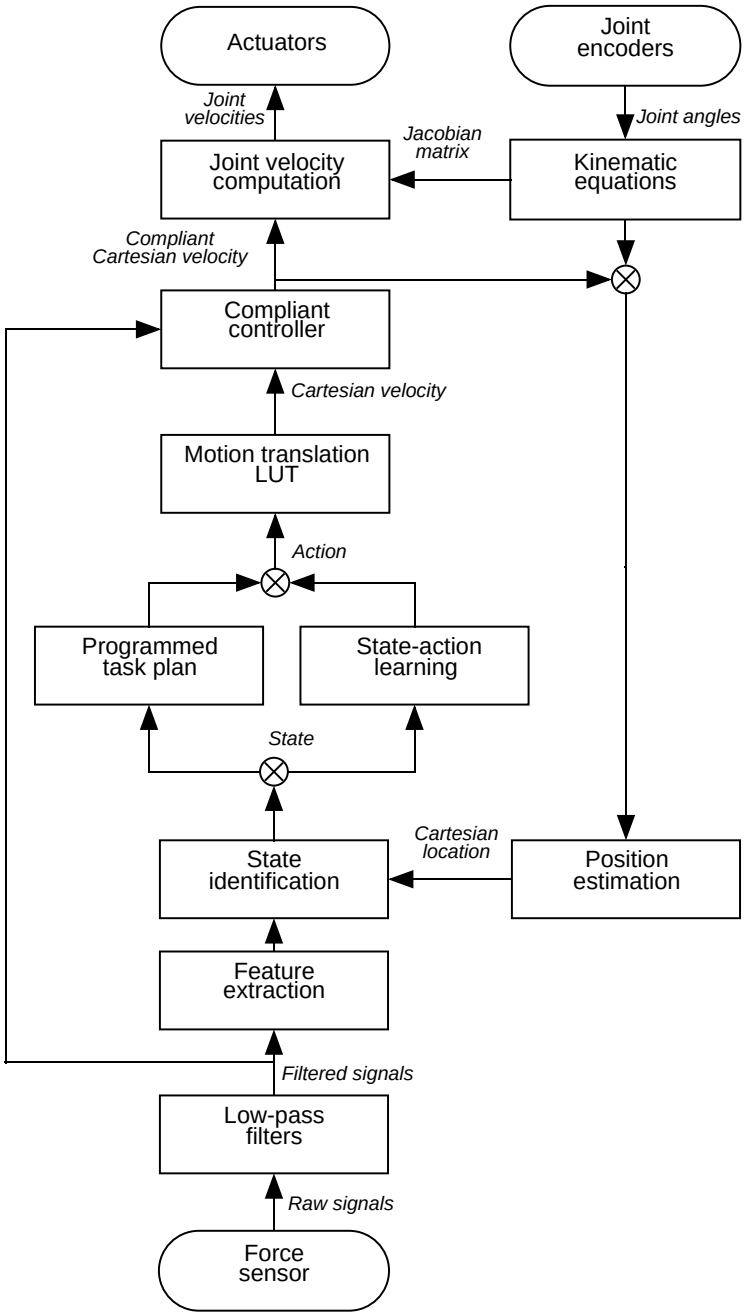


Figure 5.3: Architecture of a robotic manipulator agent

5.2.1 Specification of a fine motion task

Robot tasks involving fine motions or contact between parts and, particularly insertion and extraction, are very common in assembly and manufacturing in general. A robot arm must grasp a part, carry it to its destination and insert it adequately. This is a very error-sensitive task. Due to uncertainty, the part may be badly inserted or extracted, or even damaged. Errors are typically caused by a deficient geometric model of the environment and uncertainty in the initial pose (sensor) and control. Fine-motion is strictly a positioning problem. Perfect knowledge of the parts being manipulated and a perfect positioning device would make these tasks a trivial matter. Unfortunately, industrially manufactured parts are subject to dimensional variations; likewise the realization of a near-perfect positioning device conflicts with cost and flexibility considerations required for a general purpose robotic system.

5.2.2 Robot hardware

Sensors

Based on the specifications of a fine-motion task (small-scale space, contacts), two types of sensors are chosen: position and force. Vision is not suited for detecting small displacements, though it could be used for a first approximation.

Position sensors measure the joint angles of the robot arm. If needed, the Cartesian location of the end-effector is calculated through the direct kinematic equations of the robot. These equations are uniquely defined during the design and engineering of the arm. Examples of this class of sensors are optical encoders, resolvers, potentiometers and tachometers.

Our robot uses incremental optical encoders for measuring the relative angle of each joint with regard to an initial arbitrary zero value. This relativeness in the position produces an important additional effect in the uncertainty, since not only this uncertainty will be owed to the intrinsic defects of the arm model (gear backlash, mainly), but also to the determination of the origin configuration.

Force sensors measure forces of contact between a manipulator's end-effector and the environment which it contacts. Depending on the location of the sensing circuitry, there are two main types of force sensors: *wrist sensors* are placed between the end-effector and last joint of the manipulator (see Fig. 5.4). *Force-sensing fingers* are located at the fingertips of the end-effector, and measure from one to four components of the force acting at each fingertip.

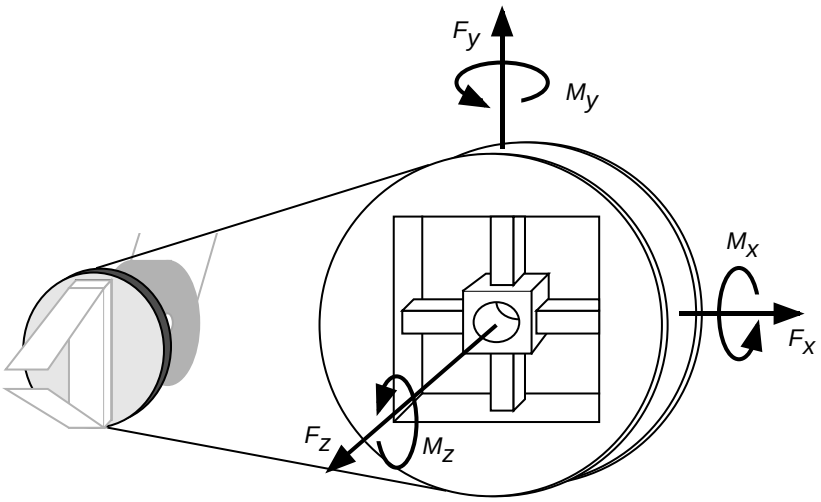


Figure 5.4: Wrist force sensor

In the robot used in the experiments, a wrist force sensor measures the six components of the force and torque vectors acting on the end-effector (Fig. 5.5 depicts the profiles of these signals during a typical assembly operation). The sensor is a cylindrical unit mounted between the wrist flange and the end-effector. It consists of 3 binding beams, each instrumented with 2 pairs of semiconductor strain gauges, totalling 6 strain measurements. As forces and moments are applied to the end-effector, these beams flex, providing a 6×1 strain measurement vector proportional to the applied 6×1 force/moment vector. The strain vector is multiplied by a 6×6 calibration matrix to produce a 6×1 force/moment vector in tool coordinates.

Actuators

Robot motions are initiated with the software, which sends commands to a motor control board. The motor control board executes closed loop control over the joint motors by reading the motor encoders and sending signals to the power amplifiers to move the motors to the commanded positions. Actuator commands are based on joint velocities. To command a Cartesian motion to the robot, a translation module is needed, which, provided with the kinematic configuration of the robot and an appropriate frame of reference, determines the motion of each joint according to the issued Cartesian motion.

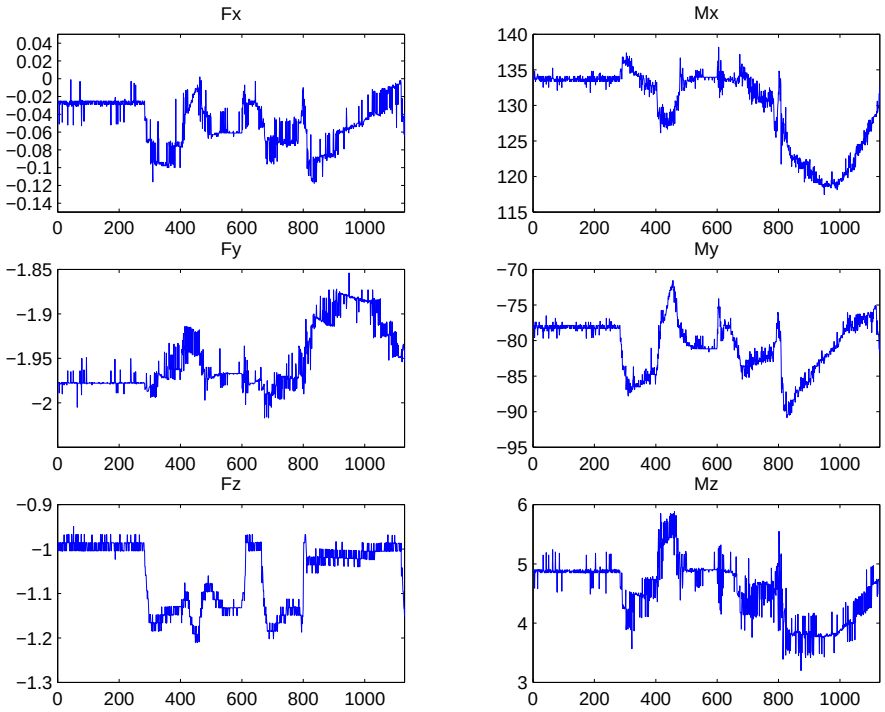


Figure 5.5: Force and torque signals

5.2.3 Signals, features and states

The information acquired from sensors is not fed into a black box, but is transformed in different stages of signal processing, feature extraction and state identification.

Signal processing

In real applications, sensor signals are corrupted with noise, which should be rejected before further processing of the signals. The presence of noise misleads other modules as to the original features of the signals. Digital filters are relatively easy to design, effective and fast to compute.

The problem of force sensing is the high frequency noise. Figure 5.6 depicts a typical force signal and its associated power spectrum density. The sampling rate of the sensor is 140 Hz. Since the dynamics of the process is much slower than this rate, it is safe to use a low-pass filter to reject the high-frequency noise without removing the interesting characteristics of the signal for the task. Following the principle of simplicity, a digital second-order Butterworth filter is chosen with a cutoff frequency at 4 Hz. The digital transfer function is:

$$H(z) = \frac{B(z)}{A(z)} = \frac{0.0071 + 0.0143z^{-1} + 0.0071z^{-2}}{1 - 1.7474z^{-1} + 0.7758z^{-2}} \tag{5.1}$$

No experiments have been carried out to compare its performance with that of other types, higher order, or different cutoff, but this does not seem critical for the task. The filtered signal is depicted Fig. 5.7, which shows that high frequencies are rejected to a great extent.

Besides noise filtering, another important operation is done at this stage. The force signal contains information about the contact forces but also the weight of the grasped workpiece, and this value depends on the orientation of the gripper. For the sake of generality, the value of the weight has to be subtracted from the sensed force to get, assuming linear superposition of forces, the real magnitude of the force due to contacts. The weight does not need to be explicitly calculated, but the system only has to read the sensor value when there is no contact, and record this value for subtraction during all the process. However, if the orientation of the gripper changes noticeably, the previous weight is not valid, and should be recorded again. The complete block diagram of the signal processing module for the force signals is depicted in Fig. 5.8. The delayed signals need to be stored in memory for the calculus of the current filtered values.

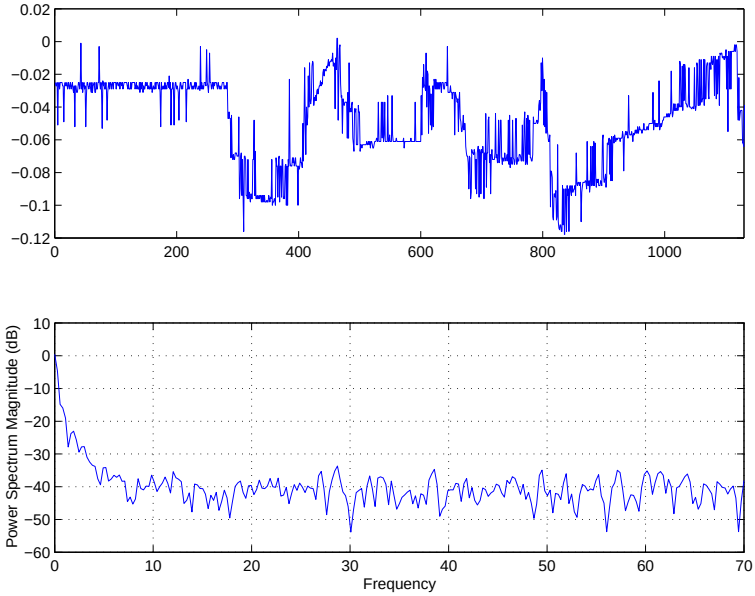


Figure 5.6: Raw force signal and power spectral density estimation

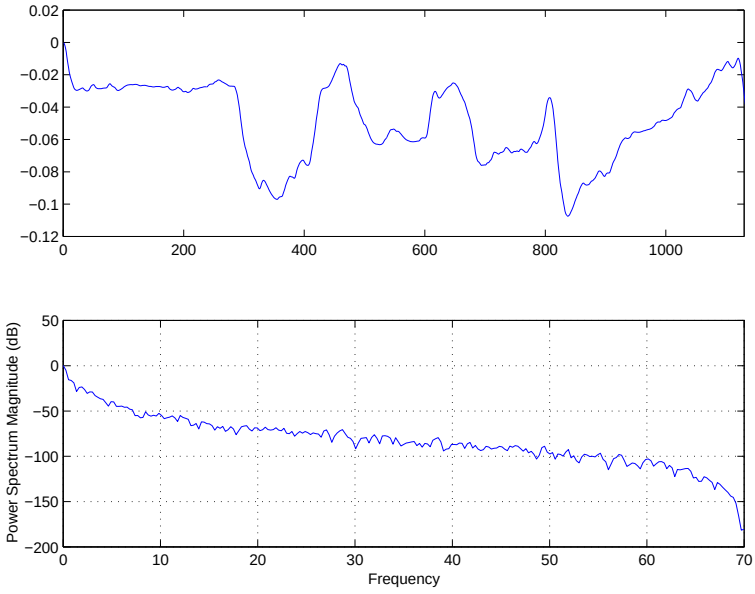


Figure 5.7: Filtered force signal and power spectral density estimation

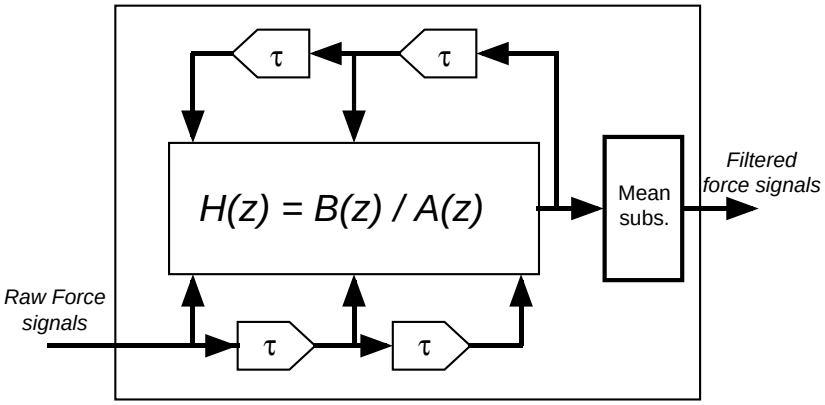


Figure 5.8: Signal processing module

Feature extraction

The selection of variables is a key problem in pattern recognition and is termed feature selection or feature extraction [Fukunaga, 1990]. Feature selection is a process of mapping the original measurements into more effective features. If the mapping is linear and if there exists a proper criterion for evaluating the effectiveness of features, we can use the techniques of linear algebra or iterative techniques to determine these mapping coefficients. Unfortunately, in many applications, there are important features which are not linear functions of the original measurements, but are highly nonlinear functions. Since there is no general algorithm to generate nonlinear mapping functions systematically, the selection of features becomes to a great extent problem-oriented.

Several criteria have been proposed for signal representation and classification purposes. In signal representation, effective features are those that represent the samples accurately. Suitable criteria are minimum mean-square error, scatter measure and population entropy. A method for selecting features based on these criteria is the Discrete Karhunen-Loéve Expansion (see details in Chap. 9 of [Fukunaga, 1990]). In classification, features that preserve class separability should be selected (Chap. 10 of [Fukunaga, 1990]). The problem in our task is that there is no class or state information available, thus suitable criteria are difficult, if not impossible, to formulate.

In the following, criteria for signal representation are used for feature extraction in the hope that they will be useful for the learning task. If a small set of features is found to represent the samples accurately, the dimensionality problem is alleviated and the state space is kept small, which in turn can accelerate the convergence of the learning process. In Chap. 4 we have

pointed out the feasibility of unsupervised learning algorithms (particularly SOMs) for extracting feature information from sensor data in robotic manipulation tasks. Since the used criterion is based on signal representation criteria, it is not guaranteed to be appropriate for the learning task.

Kohonen's self-organizing map (SOM) algorithm performs a *nonlinear projection* of the probability density function of the input space onto the two- dimensional lattice of units. The application of the SOM to the force sensor signals poses the problem of defining an Euclidean space with six dimensions of two different magnitudes (force and torque). There is a need for a scaling of the inputs, but there exists no simple rule to determine what kind of optimal rescaling should be used [Kohonen, 1995].

The practical solution adopted in the experiments is to use only the three torque signals. The reason is the strong correlation between the force and the torque, thus adding those correlated signals does not include any new information to the system. The correlation is justified by the physics of the system, and it was determined empirically. The correlation coefficient between F_x and M_y was 0.97, and between F_y and M_x it was –0.95 (see Fig. 5.5). The third component F_z is not correlated with any torque signal since it only depends on the reacting force caused by the compliant motion, but not necessarily on the point of application (i.e. type of contact).

Besides the force signals, additional position features are used by the system. It was found that, with only force information, the system was not able to learn an appropriate solution. The added position feature consists of a qualitative estimation of the relative current position with regard to the initial one. The exploration area is divided into regions, and the location features are the activation of the region where the agent is at the particular instant. The estimation is based on the accumulated motion during the trial, it is very simple to calculate and particularly effective, as its effectiveness will be demonstrated in the experiments. Dependency on the absolute location of the goal is strictly avoided, thus making possible to use the learnt knowledge directly in any other robot configuration different to the trained one.

State identification

The state is determined by the Cartesian product between the force features and the location features. Force features are processed by the SOM and produce one winner unit from all of the map units [Cervera and del Pobil, 1997a]. Location features are compared with the region borders defined on the exploration space, and only one region is selected. The state is the combination of the winner unit and the region, thus the number of states is equal to the product of the number of map units and the number of regions

of the exploration space.

The resulting state is an abstract discrete symbol, which captures all the perceptual information. In this way, a generic learning algorithm is applied to the symbolic space, not depending on the original nature of the perception signals. This approach is flexible enough to cope with changes or new type of signals, which will be adequately processed by the prior modules, and they will not require modifications on the learning procedure.

The qualitative information obtained from the feature vector is usually considered as the system state. This is true if the world is modeled as a completely observable Markovian process, where the probability of getting to the next state is fixed and it only depends on the current state. If the state of the world is observed only partially through the sensors, or it is not truly Markovian, the past information should be taken into account.

The proposed methodology considers the feature data as symbols from a regular language (which is unknown initially). The dynamic behavior of the process is approximated by a Finite State Automaton. Although the regularity assumption seems to be too restrictive, it should be noted that any language can be approximated at a desired degree with a regular language.

The language alphabet is the finite set of discrete symbols obtained from the feature vector. Since neither the states nor transitions of the automaton are known in advance, an inference process is needed to learn the structure of the automaton from example strings. This process is carried out by means of a recurrent neural network, namely an Elman network trained with the backpropagation algorithm. The FSA is extracted from the hidden layer of the network through a procedure called dynamic state partitioning. The state of the agent is determined by the state of this inferred automaton upon processing of the qualitative input (a complete example is provided in Sect. 3.5 on page 44).

5.2.4 Planning and learning

Programmed task plan

Despite the agent's capacity of learning, if any task knowledge is available it should be used at those stages where uncertainty is not important, and to guide the agent and speed up the learning process. One of the simplest plans is to perform a random walk through the state space, using any constraint derived from task knowledge that limits the exploration space. For example, is it is known that the goal is within a distance of 5 millimeters, it does not make sense to explore locations further away; the agent should be endowed with the capacity of detecting such displacement and limit the exploration

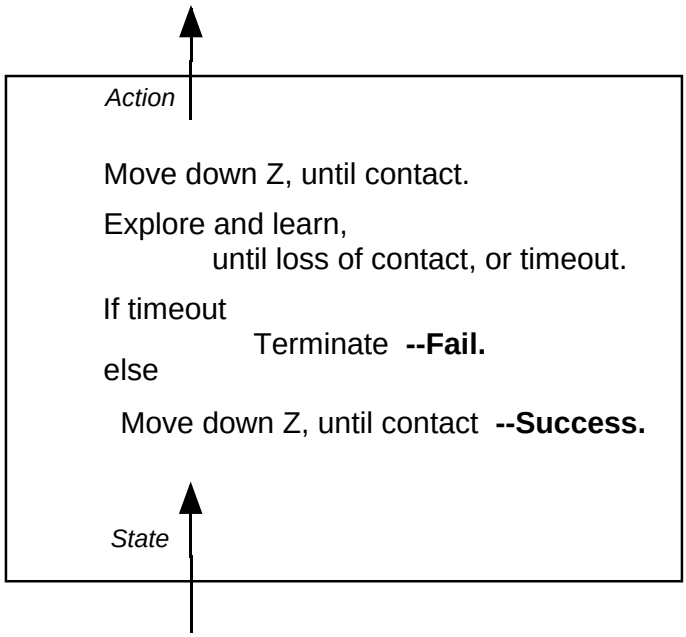


Figure 5.9: Programmed task plan

area in accordance.

Plans should explicitly capture the task knowledge. Desirable characteristics are simplicity and robustness; optimality is not a major concern, but the possibility of improving the performance through experience. Seeking from inspiration in people's way of solving tasks, a plan is a feasible solution that is gradually improved by experience and skill refinement [Cervera and del Pobil, 1997d,b].

Such an example of programmed task plan is depicted in Fig. 5.9. This plan resembles the way in which a person would perform an insertion task: just approximate to the goal and, upon failure, explore the surrounding area while gently forcing the piece into the hole; either by chance or by his/her tactile skills, the person eventually comes up with the correct insertion; if this does not occur for a long time, just finish and start again from the beginning. Formally, the plan consists of a phase when the uncertainty does not matter (move down Z), and the actual exploration phase. This is limited by a fixed uncertainty bound, which guarantees that the target is located within the bounded area.

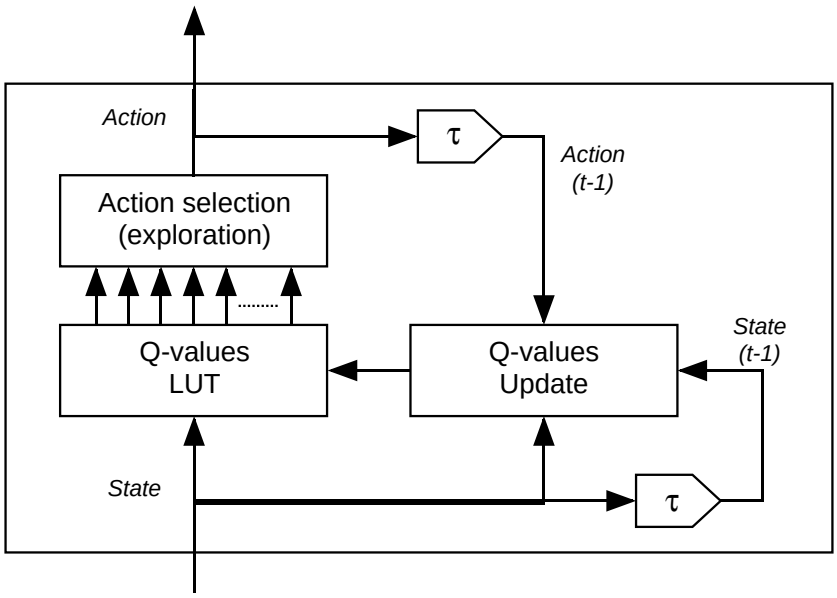


Figure 5.10: State-action learning

State-action learning

The agent's learning problem has been simplified to the extent that a reinforcement learning is directly applicable. There exists a finite set of states, a finite set of actions (chosen from the whole range of allowed directions), and only the reinforcement scheme remains undefined. A simple action-penalty representation is used, i.e. the agent is penalized (negative reinforcement) for every action it performs. Consequently, the agent will learn to solve the task with the minimum number of actions.

Since the number of steps for a trial is finite (each trial is limited by a timer; if the goal is not found within the fixed time, the current trial is aborted and a new one is started), the maximum reinforcement is bounded and there is no need to use a discount factor. Thus the reinforcement value is weighted uniformly across all the steps. Using this scheme, the Q-learning algorithm (reviewed in Sect. 3.4.2) can be directly applied to the problem.

For simplicity a look-up table (LUT) has been used to store the q-values. There are as many rows as states and as many columns as actions. Each cell stores the q-value for the corresponding pair of state and action, and represents the expected future reinforcement when such action is chosen from such state. The table is initialized with zero values, and it is updated on-line while successive trials are being executed (Fig. 5.10).

A exploration scheme based on the Boltzmann method (3.17) is used. The

temperature value, as well as the learning rate, is updated in accordance to (3.18).

5.2.5 Control algorithms

Motion translation

The learning algorithm employed in the architecture was originally developed for a discrete state and action space. Though extensions have been proposed in the literature to manage continuous state spaces, the same extension in the action space is more difficult to tackle. Discretization is usually considered a major drawback of this algorithm. Although an action is a 'primitive' in the theory, it is important to understand that, depending on the application, an action can be a high-level command that executes one of a repertoire of complex behaviors. In our experimental results, it is shown that only eight or ten different discrete actions are sufficient for successfully performing a real insertion task for cylindrical and non-cylindrical pegs. But these actions are high-level primitives, namely compliant motions Mason [1981]. Well-known algorithms are used for position control, force control, compliant motion, etc. A separate control law could be used for each distinct qualitative action.

Compliant controller

During compliant motions, the commanded velocity is modified by the sensed forces according to a programmed behavior (Fig. 5.11). Usually some degrees of freedom are position controlled, and others are force controlled. For example: when sliding the peg across the surface for exploration, the motion of the peg is position controlled along the two dimensions tangential to the surface, but the motion is force controlled along the third dimension normal to the surface. This behavior is defined by the damping matrix B, and for this simple example, the matrix is diagonal with low values for the tangential components and a high value for the normal component. Complicated compliant behaviors are possible with complex damping matrices.

Kinematic modules

The relationship between the Cartesian velocity and the velocity of each joint is defined by the Jacobian matrix. But this matrix depends on the current robot configuration, and it needs to be recalculated at each control cycle, using the robot kinematic equations.

The Cartesian location of the end-effector can be calculated in two ways: either applying the direct kinematics equations to the joint vector, or by

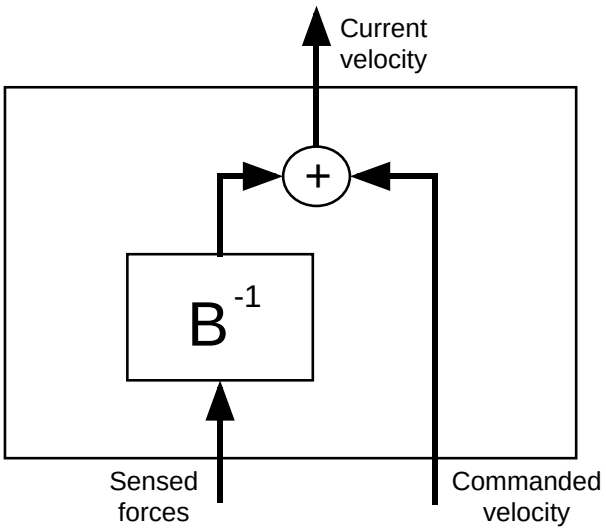


Figure 5.11: Compliant controller

adding the amount of motion to the initial location. This second approach is more efficient than the direct kinematics approach, so it is definitely used in the implementation (in a experimental comparison between the two methods there were no noticeable changes in the estimated position).

5.3 Experimental results

The system has been implemented in a Zebra Zero robot, a six d.o.f. arm equipped with a wrist-mounted force sensor (see Appendix A for details). The control cycle frequency is 140 Hz, i.e. this is the maximum frequency at which motions can be commanded to the robot actuators. This is also the sampling frequency of the force sensor. The task is the insertion of pegs of different shapes (circular and square section) into their appropriate holes. Pegs are made of wood, and the platform containing the holes is made of a synthetic resin.

Uncertainty in the position and orientation is greater than the clearance between the pegs and holes. The peg is supposed to be properly grasped by the robot gripper. The nominal goal is specified by a vector and a rotation matrix relative to an external fixed frame of reference. This location is supposed to be centered above the hole, a few millimeters upwards and well oriented, so the peg would be inserted just by moving straight along the Z axis with no rotation if there were no uncertainty present. After positioning in the nominal goal, the robot performs a guarded motion towards the hole.

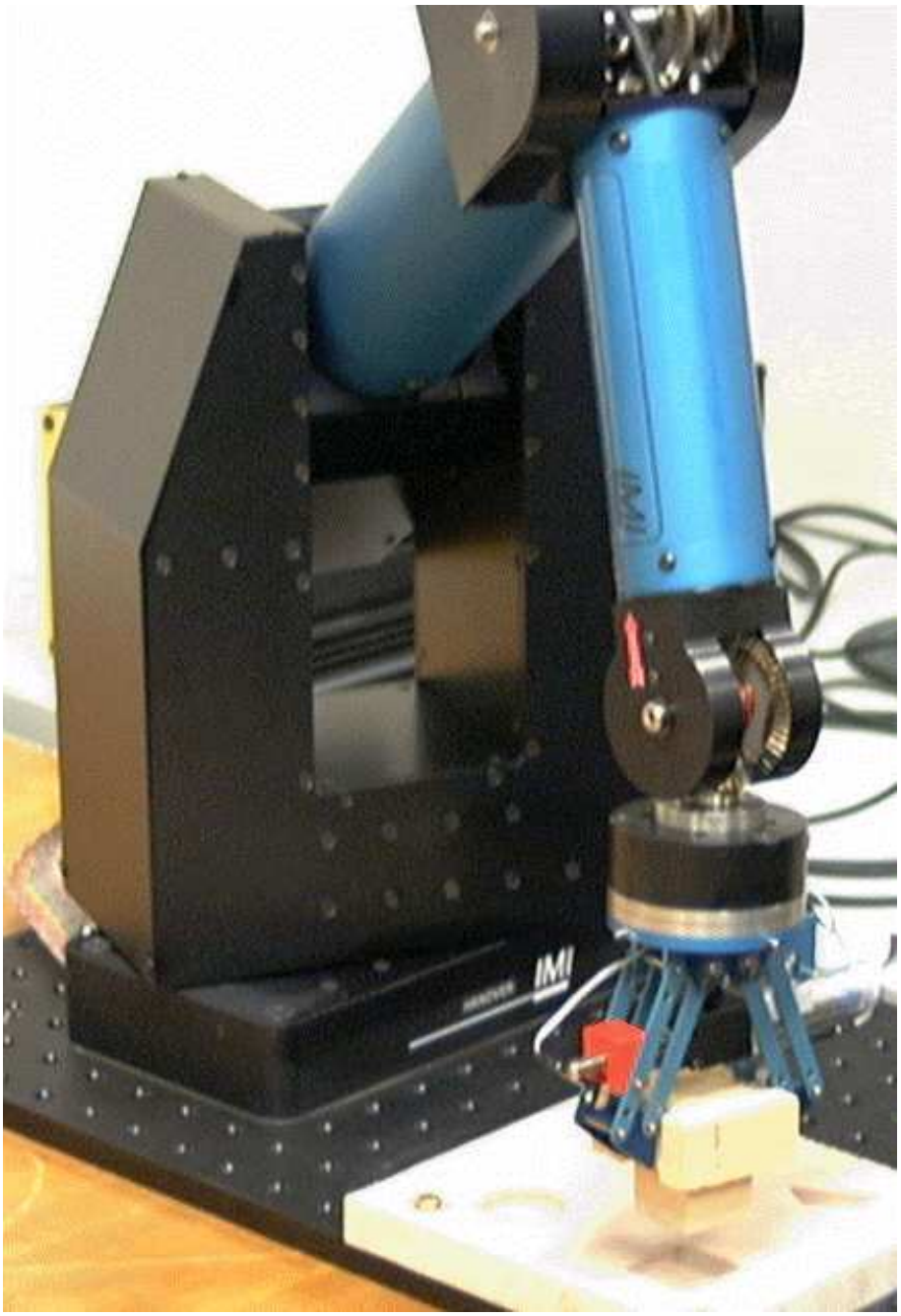


Figure 5.12: Zebra Zero robot arm, grasping a peg over the platform

The success of the operation depends on the point where the contact is detected. It is trivial that to be able to discriminate between the contact on the surface and the contact on the bottom, the depth of the hole must be greater than twice the uncertainty radius, as is usually the case.

If the insertion has failed, the robot starts a series of perception and action cycles. First, sensors are read, and a state is identified; depending on such state, an action or another is chosen, and the learning mechanism updates the internal parameters of decision. The robot performs compliant motions, i.e. it keeps the contact with the surface while moving, so it can detect the hole by a sudden force change due to the loss of contact.

Contact can be lost when moving into the hole, or due to defects of the compliant controller. The robot is commanded a new guarded motion towards the hole. If contact is detected immediately again, a false alarm is signaled and the exploration cycles are re-initiated. On the contrary, if the robot moves freely downwards before detecting a new contact, the hole has been found and the task is completed successfully.

To avoid long exploration cycles, a timeout is set which stops the process if the hole is not found within that time. In this case the robot returns to its home location, and a new trial is started.

Two types of signals are used: the force sensor signals and the relative location with respect to the initial position and orientation.

5.3.1 Case of the cylinder

The peg is 29 mm in diameter, while the hole is chamferless and 29.15 mm in diameter. The clearance between the peg and the hole is 0.075, thus the clearance ratio is 0.005. The peg has to be inserted to a depth of 10 mm into the hole. See Fig. 5.13.

The input space of the self-organizing map is defined by the three filtered torque components. The map is trained off-line with approximately 70,000 data vectors extracted from previous random trials.

Definition of the SOM

The parameters of the map are:

- Dimension: 6×4
- Topology: hexagonal
- Neighborhood: bubble

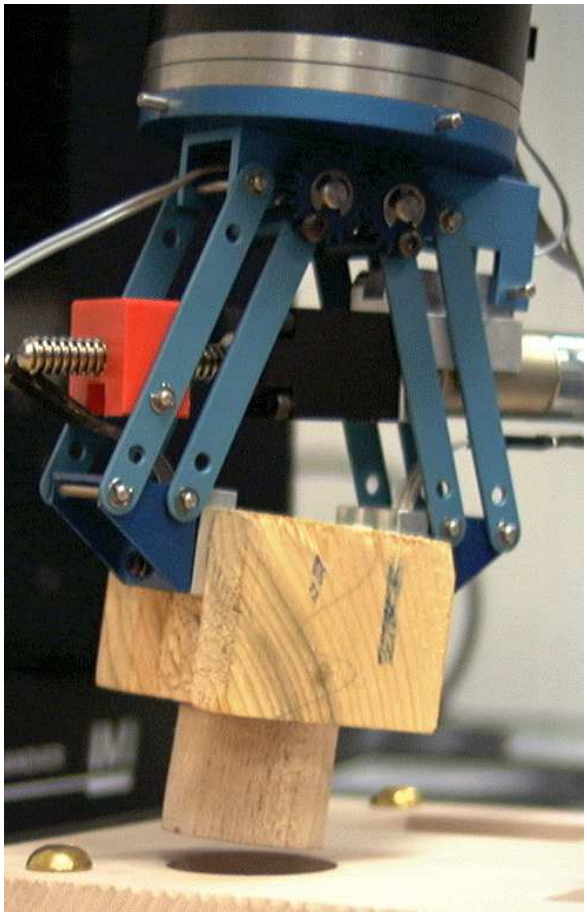


Figure 5.13: Robot gripper grasping the cylinder peg

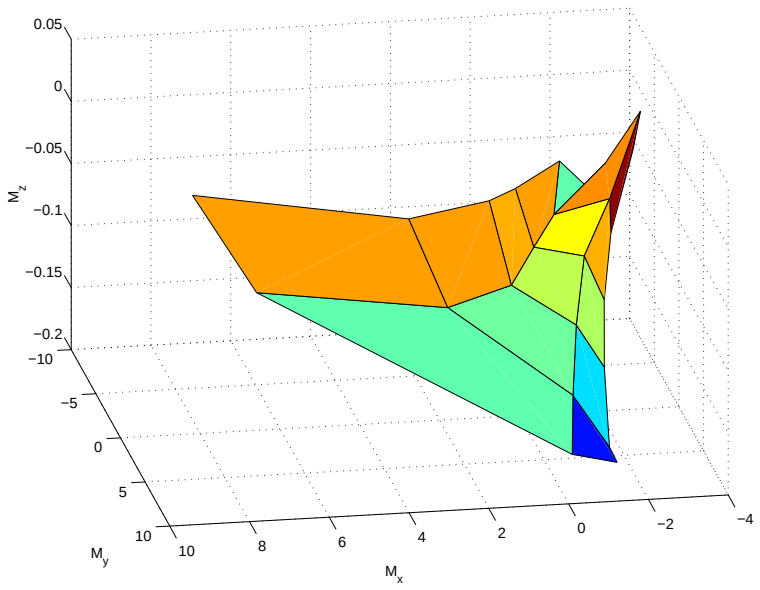


Figure 5.14: Self-organizing map in the input space

Following the recommendations given by Kohonen [1995] for good convergence of SOMs, the map is trained in two phases:

- 150,000 iterations, learning rate = 0.1, radius = 4
- 1,500,000 iterations, learning rate = 0.05, radius = 2

Due to the low dimensionality of the input space, it is possible to visualize the trained map in a plot (Fig. 5.14). The map is also projected in the two most significant dimensions (M_x, M_y). The partition of the signal space corresponds to the Voronoi diagram defined by the map units, whose projection onto M_x and M_y is depicted in Fig. 5.15.

Learning procedure

The agent is trained with a sequence of trials, each of which starts at a random position within an uncertainty radius of 3 mm. To absolutely ensure that the goal is within the exploration area, this area is set to a 5 mm square, centered at the real starting position. Exploration motions are tangential to the surface, i.e. along the X and Y dimensions. The exploration space is partitioned in nine regions with thresholds in -2 and +2 for both X and Y; each of these regions define a *qualitative location state*. The state is determined by combining the winner unit of the map and this relative qualitative

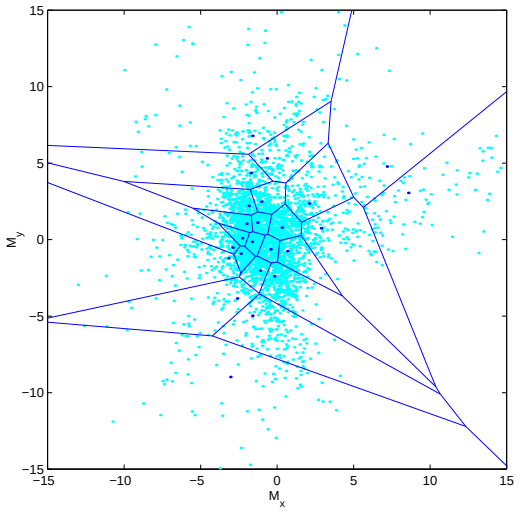


Figure 5.15: Voronoi diagram defined by the SOM on dimensions (M_x, M_y)

position with respect to the initial location, thus the total number of states is $24 \times 9 = 216$.

Contact is detected by two thresholds in the force component F_z (normal to the surface): a contact is gained when F_z is less than -0.1 Kgf, and the contact is lost if F_z is greater than -0.05 Kgf. This dual-threshold method overcomes small variations in the reaction force due to friction, the irregular surface, and the compliant controller. During compliant motions, a force F_z equal to -0.15 Kgf is constantly exerted on the surface.

The action space is discretized. Exploratory motions consist of fixed steps in eight different direction of the XY-plane (0, 45, 90, 135, 180, 225, 270 and 315 degrees). These motions are hybrid, with some degrees of freedom (XY) being position-controlled and other degrees (Z) being force-controlled, as stated in Mason [1981]. The complexity of the motion is transferred to the control modules, and the learning process is simplified.

Each learning step consists of a sensor reading and an action. During processing, the robot stops the last motion and senses the forces during 55 control cycles (approximately 55/140 seconds). The reason for stopping motion during sensing is to alleviate the effect of dynamic friction, since in this setup it completely misleads the learning algorithm about the contact forces.

The sensed forces and the current relative location are used to determine the current state. An action is chosen based on the Boltzmann exploration

scheme (3.17). Initially, random actions are chosen; as the temperature decreases, the action with higher value for the current state is chosen more often. The action causes a compliant motion to be commanded during 70 control steps (0.5 seconds), but it might finish before if the contact is lost (i.e. the hole is found). Between the action and the next sensing step there is a pause period of 45 cycles (about 45/140 seconds) to allow the motion to stop completely.

Learning results

The learning update step consists of modifying the q-value of the previous state and the performed action according to the reinforcement and the value of the next state. The agent receives a constant negative reinforcement for each action it performs (action-penalty representation). The best policy, the one that maximizes the obtained reinforcement, is that which achieves the goal with the minimum number of actions. Experimental results are shown in Fig. 5.16. The critical phase is the surface-compliant motion towards the hole. The system must learn to find the hole based on sensory information. The exploration time of 4,000 consecutive trials is shown. The timeout is set to 20 seconds in each trial. The smoothed curve was obtained by filtering the data using a moving-average window of 100 consecutive values. The learning algorithm is executed since the beginning. Temperature is gradually decreased from the initial value of 75, thus progressively turning down exploration.

Figure 5.17 depicts the probability of successful insertions over time. This probability is estimated by calculating the percentage of achieved goals during 100 consecutive trials. The system evolves from a bare 38% of successful insertions during the first 500 trials (accomplished by random motions) to a satisfactory 93% of success during the last 500 trials of the learning process.

After 1,500 trials, the insertion time is considerably improved over the values at the first steps. Despite small oscillations, the time converges to this value. Although the results presented by Gullapalli et al. [1994] show a faster convergence for a similar task, one should note that the setup is quite different, since he uses the real location as input to the robot. It is unclear how the trained system could generalize to a different test location. Our system uses relative forces and a relative open-loop estimation of the location of the end-effector. Theoretically, this information is invariant with respect to the position and orientation of the target. Any goal should be achieved with equal probability, provided an initial location within a given bound of the real goal. Our system has been tested on different locations, showing a good performance (83% of successful insertions) when there is no

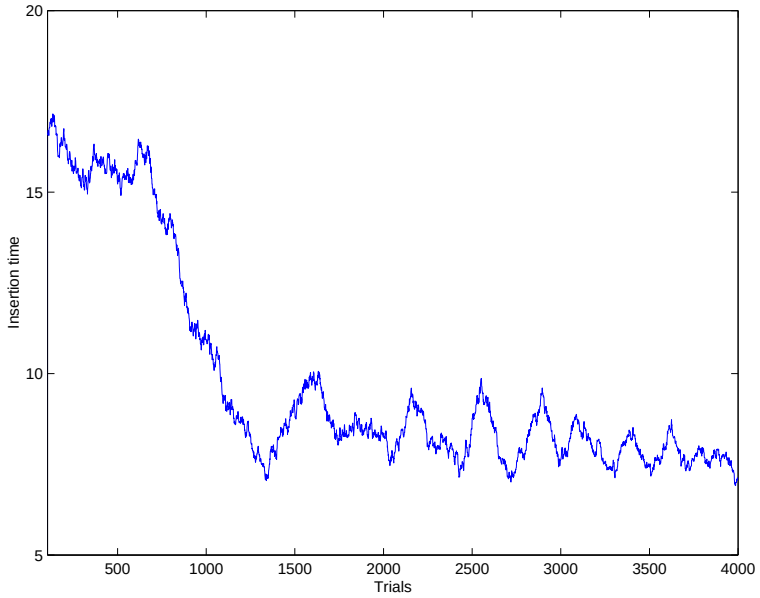


Figure 5.16: Smoothed insertion time taken on 4,000 trials of the cylinder task

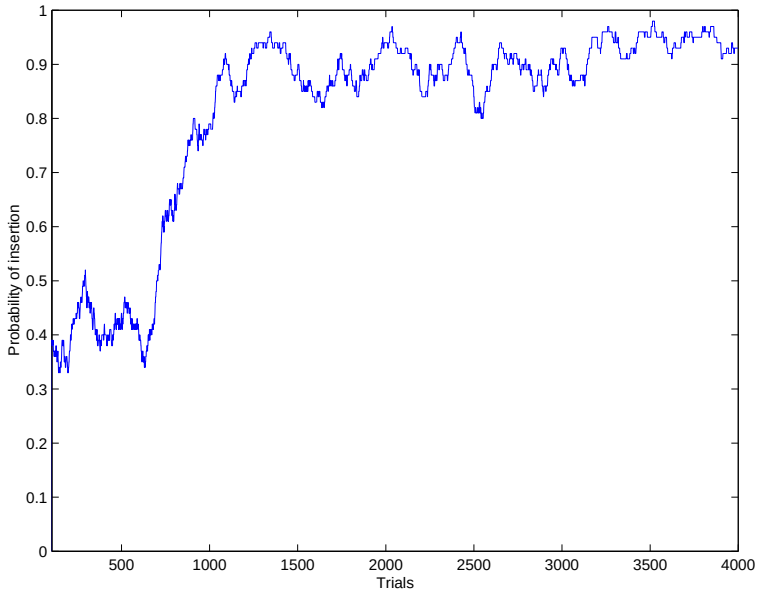


Figure 5.17: Evolution of the probability of successful insertion during the training process

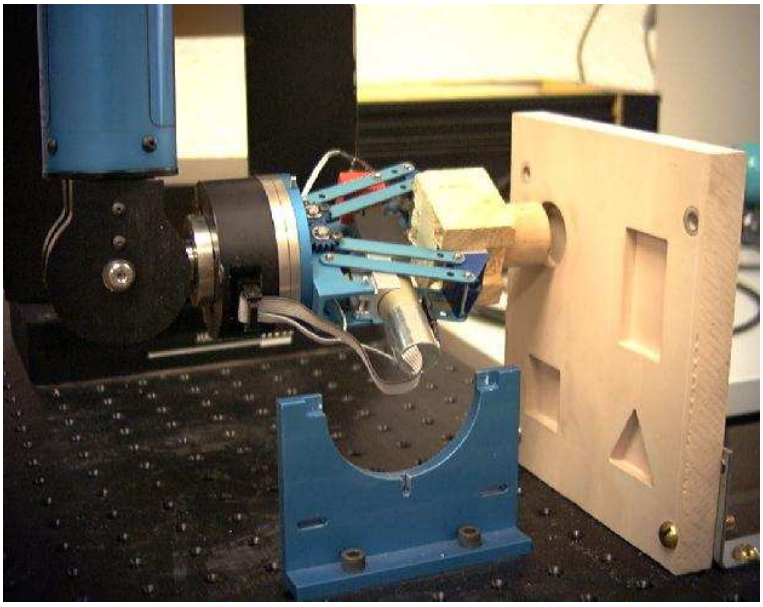


Figure 5.18: New orientation of the hole

orientation change.

However, if the hole is rotated 90 degrees (see Fig. 5.18), there is a significant loss in performance (only 44% of insertions are successful), but, upon additional training, the system quickly recovers a near perfect performance for the new setup (Fig. 5.19 shows that with less than 500 new trials, more than 70% of insertions are successful, whereas during the training process, 1000 trials were required to achieve this rate). Since all the measurements are relative, one would expect better results. The best solution would be to interleave different positions and orientations during the training process, but this is not possible in our setup.

Since the trial timeout is set to 20 seconds, additional experiments were carried out with a higher timeout in order to study the distribution of successes over a long time, and compare the differences between the random and learning strategies. Figure 5.20 depicts the distribution of successful insertions with respect to time, for 1,000 random trials and 1,000 trials using the learnt controller (but learning has been turned off during these trials). As expected, it was found out that it is possible to achieve nearly all the insertions with random motions, provided the necessary amount of time, but the learnt controller achieves best results with significantly less time.

Experimental results have shown that the system learns incrementally from an initially random strategy, by improving its performance based only on its own experience. A significant amount of task knowledge is embedded

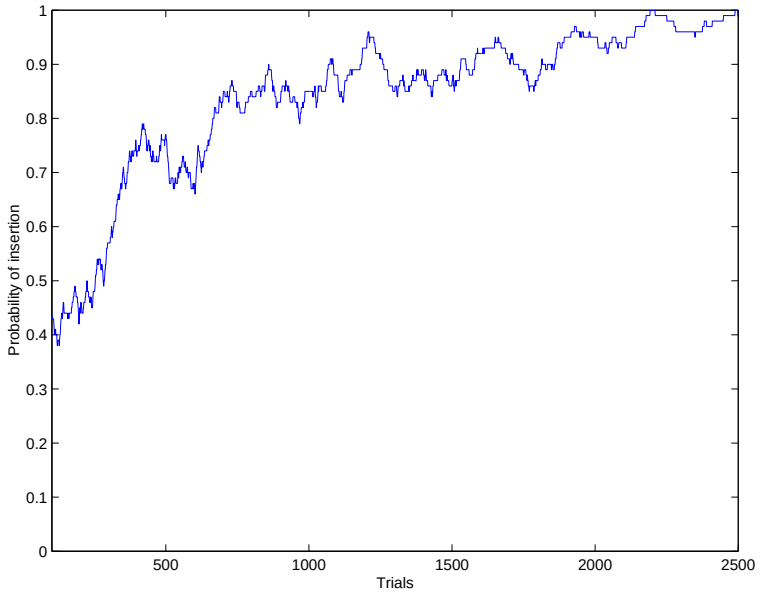


Figure 5.19: Adaptation to a new hole setup

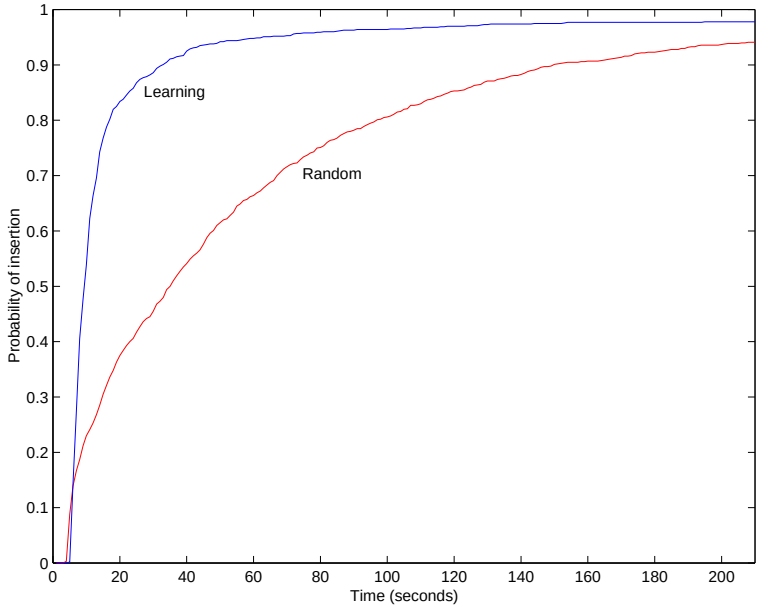


Figure 5.20: Probability of insertion for random and learned strategies

in the system architecture, which simplifies the learning problem. However, the imprecision, noise and inherent difficulties of a real robot are dealt with a discrete learning algorithm.

5.3.2 Case of the square peg

Due to its radial symmetry, the cylinder is more simple than other pieces for insertions. It has been widely studied in the literature since the force analysis can be done in two dimensions. Analytical results for pegs of other shapes are much more difficult: Caine et al. [1989] developed an heuristic approach to manage 1,000 different contact states of a rectangular peg insertion.

In our architecture, it is very simple to extend the agent's capabilities to deal with other shapes than the cylinder. Besides the uncertainty in the position along the dimensions X and Y (tangential to the surface), the agent must deal with the uncertainty in the orientation with respect to the Z axis (the hole axis, which is normal to the surface). In addition to the center of the square being located exactly in the center of the hole, it has to be exactly oriented to allow a proper insertion. The peg used in the experiments has a square section, its side being 28.8 mm. The hole is a 29.2 mm square, thus the clearance is 0.2 mm, and the clearance ratio is approximately 0.013. The peg is made of wood, like the cylinder, and the hole is located in the same platform as before (Fig. 5.21).

The radius of uncertainty in the position is 3 mm, and the uncertainty in the orientation is ± 8.5 degrees. The exploration area is a 5 mm square and an angle of ± 14 degrees. The area is partitioned in 9 regions, and the angle is divided in three segments. The self-organizing map contains 6×4 units, likewise the previous case. The rest of the training parameters are the same as before.

Again, the input space of the SOM is defined by the three filtered torque components. The map is trained off-line with approximately 70,000 data vectors extracted from previous random trials.

Definition of the SOM

The parameters of the map are:

- Dimension: 6×4
- Topology: hexagonal
- Neighborhood: bubble

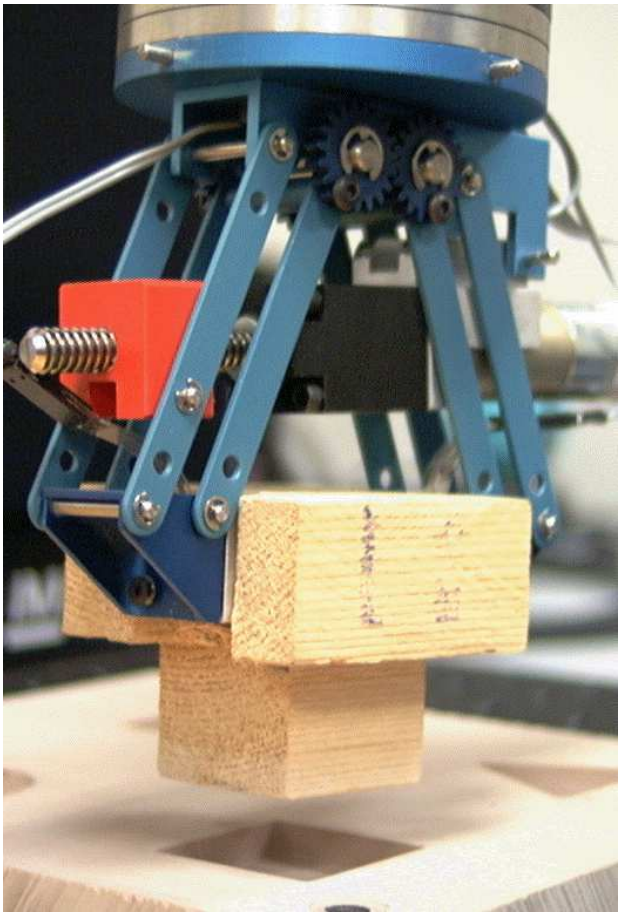


Figure 5.21: Robot gripper grasping the square peg

Following the recommendations given by Kohonen [1995] for good convergence of SOMs, the map is trained in two phases:

- 150,000 iterations, learning rate = 0.1, radius = 4
- 1,500,000 iterations, learning rate = 0.05, radius = 2

Due to the low dimensionality of the input space, it is possible to visualize the trained map in a plot (Fig. 5.22). The map is also projected in the two most significant dimensions (M_x, M_y). The partition of the signal space corresponds to the Voronoi diagram defined by the map units, whose projection onto M_x and M_y is depicted in Fig. 5.23.

The total number of states is $27 \times 24 = 648$. Though this is the total number of states, some of them may actually be never visited at all, thus the number of real states is somewhat smaller. There is a tradeoff between the number of states and the learning speed. If more states are used, the internal representation of the task is more detailed, but more trials are needed to learn the q-values of all those states. With less states, the q-values are updated more quickly and the learning process is faster. Unfortunately, there is no general method for selecting the number of states, and it becomes a task-dependent heuristic process.

Two new actions are added, namely rotations around the normal axis to the surface, since symmetry around it does not hold any more. A qualitative measure of that angle is also included in the agent's location estimation. Since 10 different actions are possible at each state, the table of q-values has 6,480 entries.

The rest of the architecture and the training procedure remains unchanged. The increased difficulty of the task is shown by the low percentage of successful insertions that are achieved randomly at the beginning of the learning process. Only about 15% of the insertions are performed in less than 20 seconds time. The higher difficulty is the reason for the longer learning time and the worst performance achieved with respect to the cylinder.

Learning results

Figure 5.24 depicts the insertion time during 8,000 learning trials. One should take into account that any failed insertion is rated at 30 seconds, which is not true. The agent's improvement is shown more clearly in Fig. 5.25, which depicts the probability of successful insertion within 30 seconds time. The process is slightly more unstable than the cylinder due to the increased difficulty, but the agent achieves a significant 80% of successful insertions. If this timeout is not considered, the benefit is more apparent. Figure 5.26

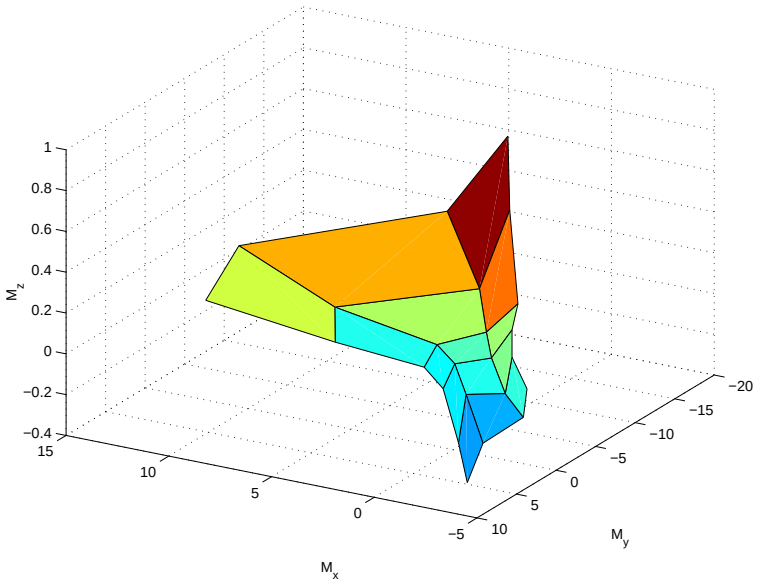


Figure 5.22: Self-organizing map in the input space for the cube task

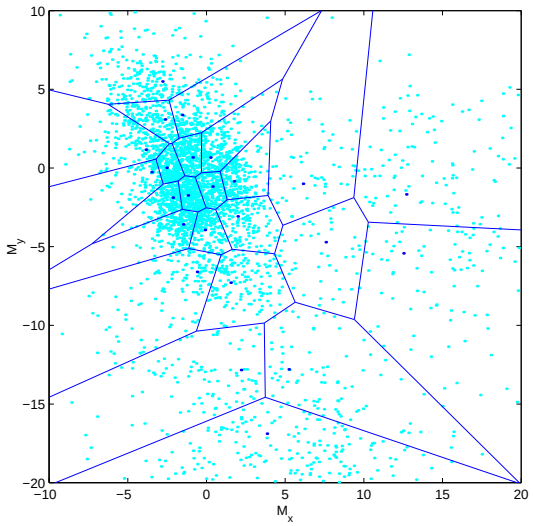


Figure 5.23: Voronoi diagram defined by the SOM on dimensions (M_x, M_y), for the cube task

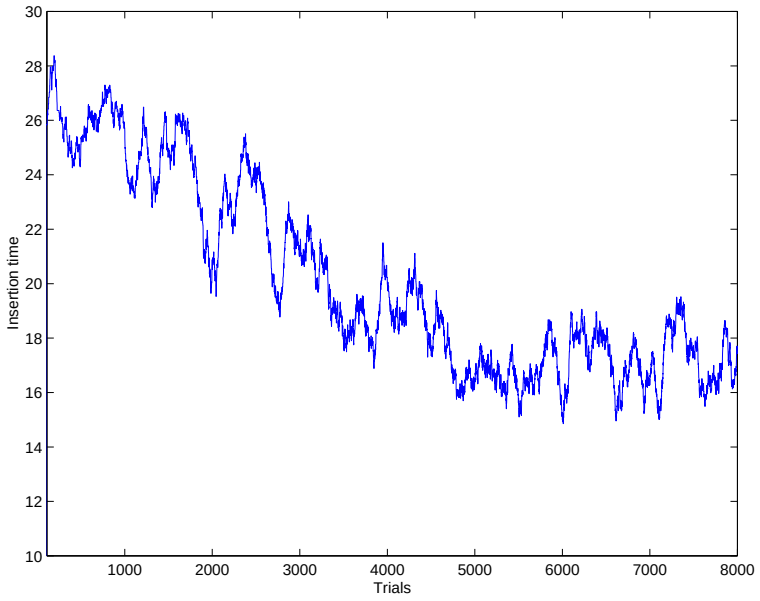


Figure 5.24: Smoothed insertion time taken on 8,000 trials of the cube task

depicts the probability of successful insertion for 1,000 random trials and 1,000 trials with the learnt controller, with respect to a time up to 210 seconds (3 and a half minutes). The difference is more dramatic than in the case of the cylinder, since the random controller, even for a long time, is capable of performing a low percentage of trials (about 45%), whereas the learnt controller achieves more than 90% of the trials.

As far as we know, this is the best performance achieved for this task using a square peg. In [Gullapalli et al., 1994] only results for the cylinder are presented and, though generalizing to other shapes is said to be possible, no real experiments are carried out.

5.3.3 Case of the triangle peg

The architecture is not restricted to square shapes, but in principle it can be used with any non-symmetric shape. However, concave shapes have not been experimented. Results are now presented for a triangle peg, with three equal edges (see Fig. 5.27). Each edge is 30.5 mm long, and the hole edges are 30.9 mm long.

The representation of the state space is exactly the same that has been used for the square.

As before, the radius of uncertainty in the position is 3 mm, and the

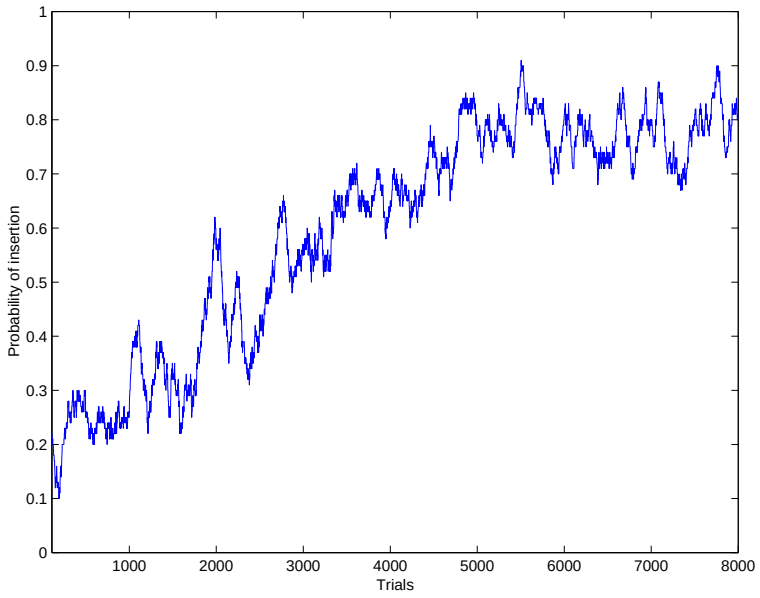


Figure 5.25: Evolution of the probability of successful insertion during the training process

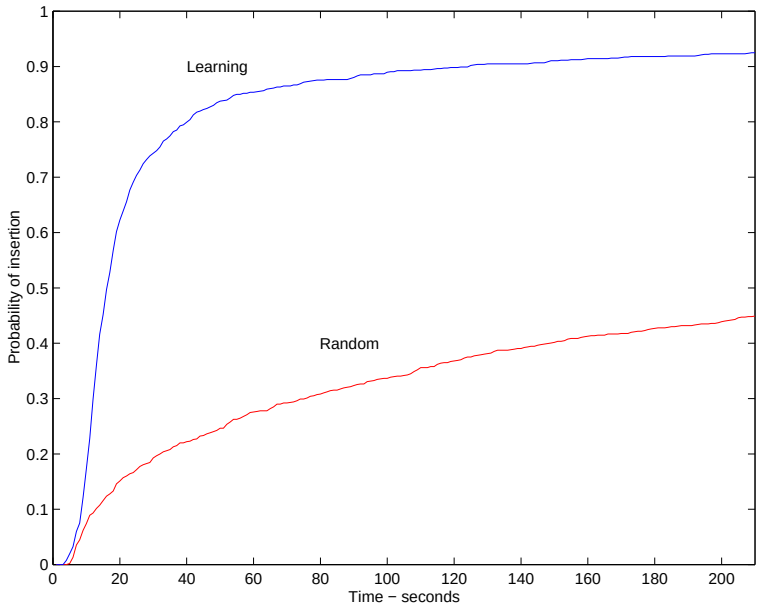


Figure 5.26: Probability of insertion for random and learned strategies

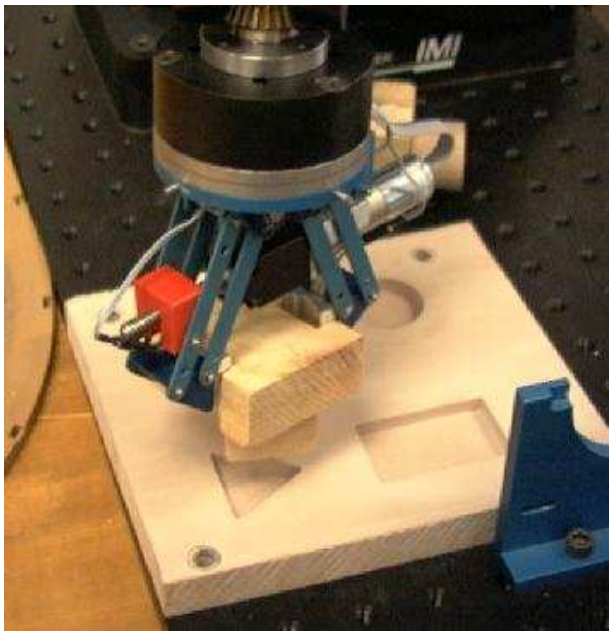


Figure 5.27: Robot gripper grasping the triangle peg

uncertainty in the orientation is ±8.5 degrees. The exploration area is a 5 mm square and an angle of ±14 degrees. The area is partitioned in 9 regions, and the angle is divided in three segments. The self-organizing map contains 6×4 units. The rest of the training parameters are the same as before too.

The total number of states is $27 \times 24 = 648$ and the same actions (8 translations and 2 rotations) are used.

As in the two previous cases, the input space of the SOM is defined by the three filtered torque components. The map is trained off-line with approximately 70,000 data vectors extracted from previous random trials.

Definition of the SOM

The parameters of the map are:

- Dimension: 6×4
- Topology: hexagonal
- Neighborhood: bubble

Following the recommendations given by Kohonen [1995] for good convergence of SOMs, the map is trained in two phases:

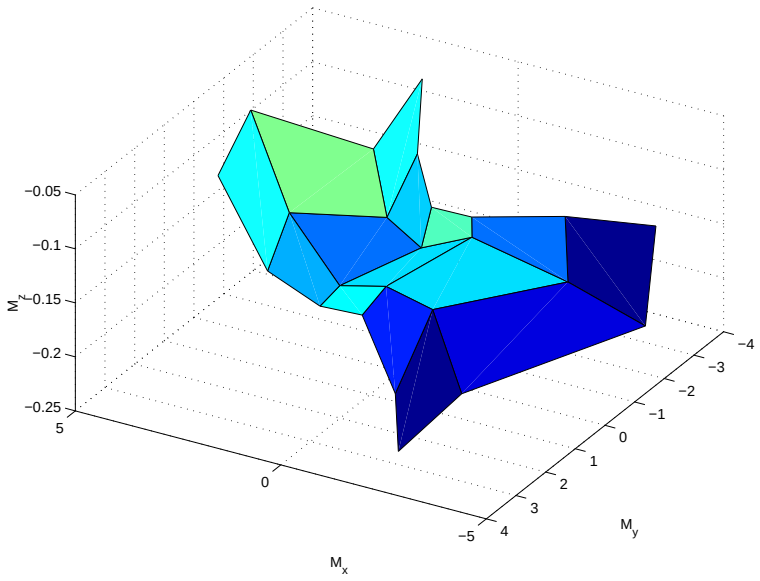


Figure 5.28: Self-organizing map in the input space for the triangle task

- 150,000 iterations, learning rate = 0.1, radius = 4
- 1,500,000 iterations, learning rate = 0.05, radius = 2

Due to the low dimensionality of the input space, it is possible to visualize the trained map in a plot (Fig. 5.28). The map is also projected in the two most significant dimensions (M_x, M_y). The partition of the signal space corresponds to the Voronoi diagram defined by the map units, whose projection onto M_x and M_y is depicted in Fig. 5.29.

Learning results

The evolution of the mean insertion time during 8,000 learning trials is depicted in Fig. 5.30. The improvement is not as apparent as in the previous cases. Moreover, the probability of insertion (Fig. 5.31) only reaches about 60% of success after the training process, whereas 80% of successful insertions were attained in the cube example.

This is quite surprising, since initially the probability of insertion for the triangle is higher, and that means that it is easier to insert the triangle *randomly* than the cube. However, it is more difficult to improve these skills based on the sensed forces for the triangle. This could be caused by the different contact states, which might be more informative in the case of the cube. This is not a contradiction at all. Possibly, the contacts of the triangle

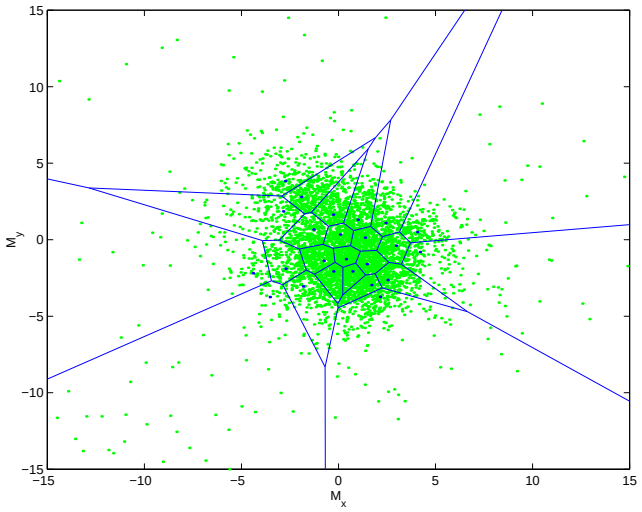


Figure 5.29: Voronoi diagram defined by the SOM on dimensions (M_x, M_y), for the triangle task

are more ambiguous, thus making it difficult to learn a good strategy for the insertion task.

Figure 5.32 depicts the probability of successful insertion for 1,000 random trials and 1,000 trials with the learnt controller, with respect to a time up to 210 seconds (3 and a half minutes). This picture clearly shows that the random strategy achieves more successful strategies than in the case of the cube, but the learned strategy is not as good as that learned in the previous case.

Unfortunately since there are no other published works for a similar task, these results cannot be compared to test if our hypothesis is true. Nevertheless, this absence of results in the literature might be indicative of the difficulties for properly learning this task.

Learning using the previous SOM

An interesting generalization test is to use a SOM trained with samples from insertions of the square peg for learning the insertions of the triangle peg. Though trained with different shapes, the purpose is to test if the features learnt with the square are useful for the insertion of other shapes. Since the size of the SOMs is the same, the state representation is not modified at all.

The evolution of the mean insertion time during 8,000 learning trials is depicted in Fig. 5.33. The results are very similar to those obtained before

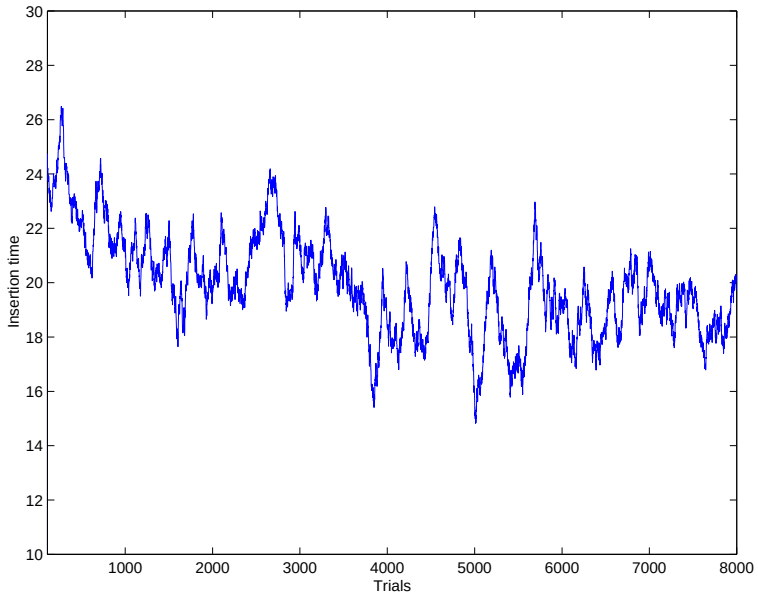


Figure 5.30: Smoothed insertion time taken on 8,000 trials of the triangle task

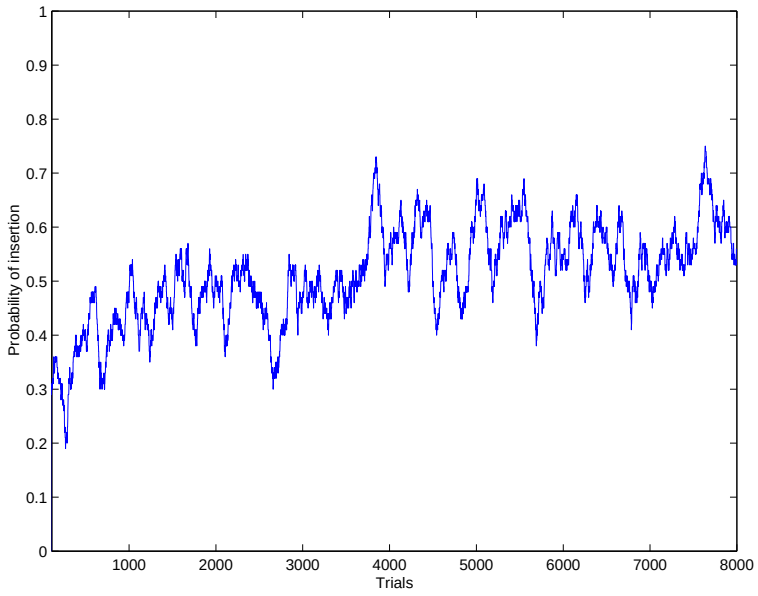


Figure 5.31: Evolution of the probability of successful insertion during the training process

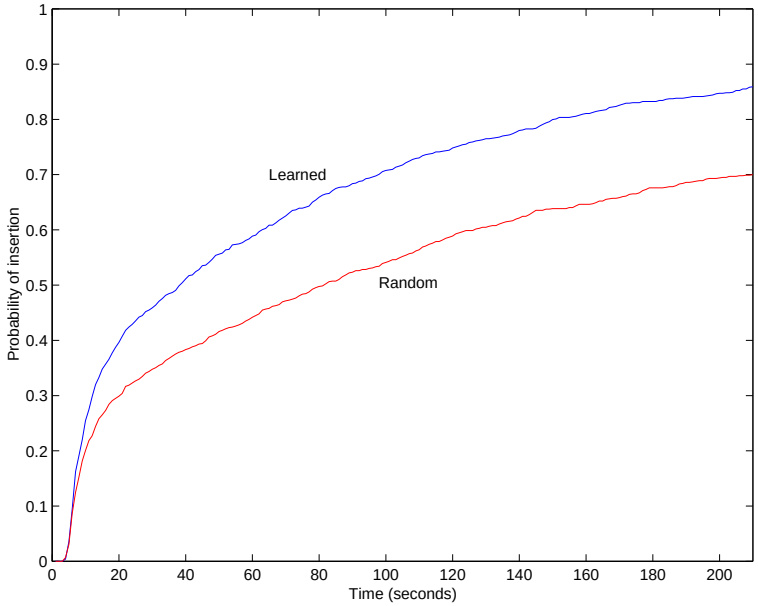


Figure 5.32: Probability of insertion for random and learned strategies

with a specific SOM. The probability of insertion is depicted in Fig. 5.34.

Figure 5.35 depicts the probability of successful insertion for 1,000 random trials, 1,000 trials with the strategy learnt with the specific SOM, and 1,000 trials with the strategy learnt with the SOM from the cube task, with respect to a time up to 210 seconds (3 and a half minutes).

Surprisingly enough, results with the cube SOM are slightly better than those obtained with the specific SOM. A possible explanation is that the SOM trained with the cube is *more powerful* than that trained with the triangle. By examining Figs. 5.23 and 5.29, which depict the Voronoi diagrams of both SOMs, one can see that the cube SOM is covering a wider area of the input space than the other one. It might occur that some input data has not much influence during the training process of the triangle SOM (due to its low probability density) but is rather important for the learning of the insertion strategy. Since the cube SOM is covering a wider area, maybe some states are properly identified with this SOM whereas they are ambiguous with the triangle SOM.

This is an interesting result which demonstrates the generalization capabilities of the SOM for extracting features which are suitable for different tasks.

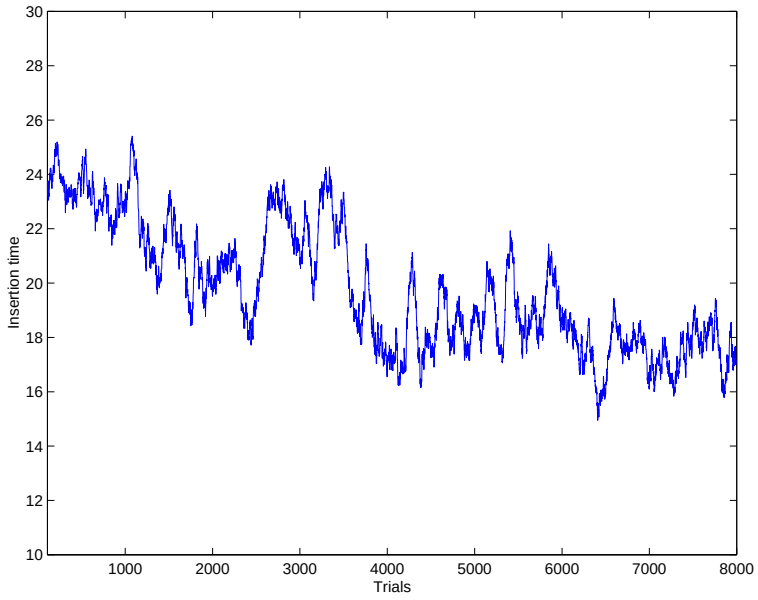


Figure 5.33: Smoothed insertion time taken on 8,000 trials of the triangle task, with the SOM trained with the square peg

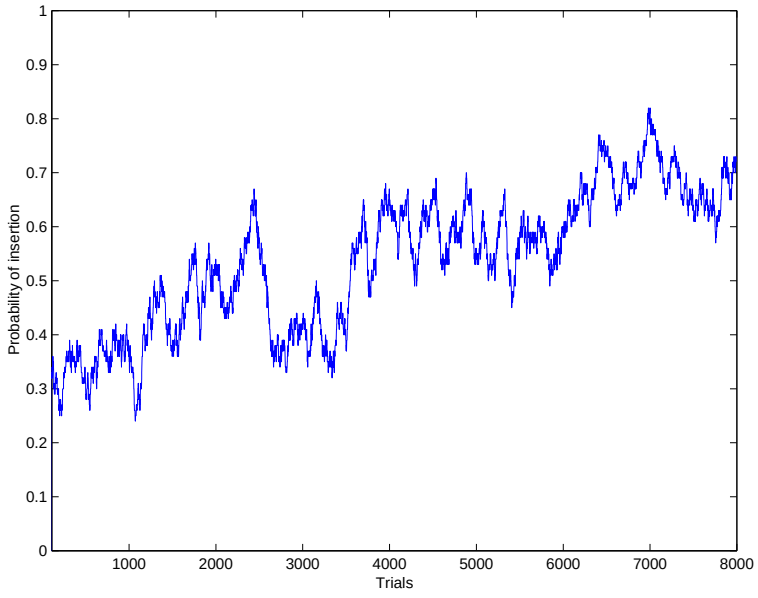


Figure 5.34: Evolution of the probability of successful insertion during the training process, with the SOM trained with the square peg

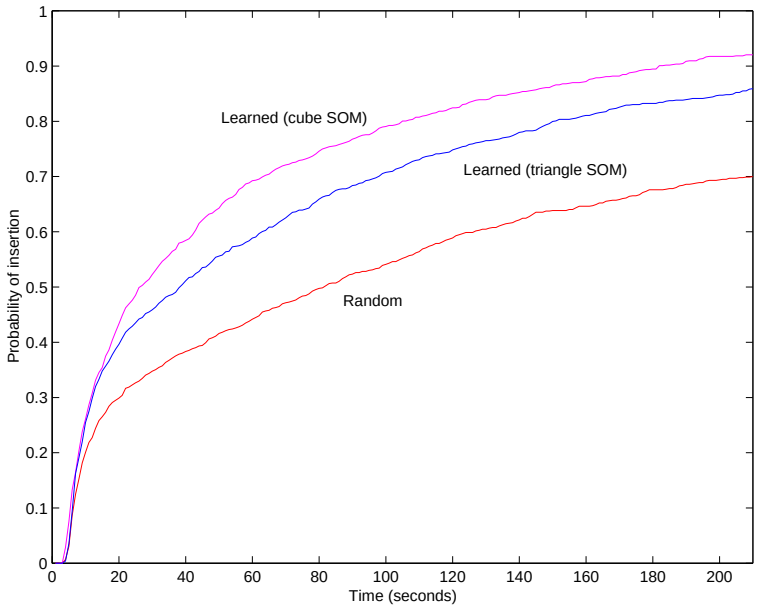


Figure 5.35: Probability of insertion for random and learned strategies with SOMs trained with the triangle and the cube

Chapter 6

Conclusions and future work

6.1 Main contributions of this thesis

The main target of this thesis has been successfully achieved: a real robot arm is capable of learning a complex insertion task based on its own experience through perception and action. In the way to the goal, some significant results have been achieved regarding new aspects of self-organizing maps and Q-learning, as well as new contributions to the theory of neural networks like an algorithm for supervised learning with SOMs and enhancing reinforcement learning with recurrent networks.

Throughout this thesis, we have emphasized the *engineering goal* of artificial intelligence: there was a real problem, which has been solved with a number of available tools in the fields of neural networks and machine learning, whose capabilities have been investigated with the target problem of robotic fine motion in mind. The integration of several tools, backed by simulations and examples, and their implementation on a real robot to solve the initial problem are the contributions of the thesis to the field of robotic intelligence.

Below, the main contributions of this thesis are summarized:

- **A supervised learning algorithm with SOMs**. This hybrid method has been described in Sect. 3.2 on page 27 and a number of examples with synthetic and real data of different available benchmarks have been presented.

- **Enhancement of Q-learning with recurrent networks**. It has been shown how the inference of finite state automata with recurrent networks enhances the learning power of Q-learning. Section 3.5 on page 44 presented a simulation of a sensor-based task, which turned

out to be impossible to solve if past history was not taken into account. An Elman network, trained to predict the next sensing symbol, learns a representation of the finite state automaton which allows the reinforcement learning algorithm to succeed in finding a solution.

- **Monitoring fine motion tasks with SOMs**. Simulations of two-dimensional assemblies and the experiments in a flexible manufacturing cell have been carried out for the study of the SOM behavior with signals generated from such tasks throughout Chap. 4. Through unsupervised learning, the features extracted from the map turn out to be significant with regard to the contact state of the task. The map is a valuable tool for monitoring of the assembly process, fault detection and error recovery. It is used later in the development of a **sensor-based qualitative plan** for insertion tasks in two dimensions, which is implemented on simulations.

- **Q-learning states with SOMs**. Self-organizing maps are useful for extracting features from input signals. Selection of winner units is a natural way of discretizing the input space into a finite set of states. These states (combined with other sources of information) are directly usable by the reinforcement learning algorithm (with a look-up table representation) for learning the association of states and actions in order to solve the fine motion insertion task (Sects. 5.2.3 on page 131, and 5.3 on page 137).

- **A robot learning architecture for manipulation**. Unsupervised learning of input features and reinforcement learning of action policies form the basis of this architecture, though a number of parameters need to be fixed by the operator (size of SOM, Q-learning parameters, state resolution). The system achieves position-independent skills in fine motion assembly tasks, and thorough experiments are reported in Sect. 5.3. Peg insertions with uncertainty are learnt within a reasonable amount of time. Moreover, the insertion of non-cylindrical parts with position uncertainty is beyond the state-of-the-art frontier in fine motion planning.

6.2 List of selected publications

The work carried out in this thesis has given rise to a number of publications. A selected list is presented below.

- International journals

  - Cervera, E., del Pobil, A. P., Marta, E., Serna, M. A. (1996). Perception-based learning for motion in contact in task planning. *Journal of Intelligent and Robotic Systems*, **17**, pages 283–308, Kluwer Academic Publishers.
  - Cervera, E., del Pobil, A. P. (1997c). Multiple self-organizing maps: a hybrid learning approach. *Neurocomputing* (in press). Elsevier Science.

- International book chapters

  - Cervera, E., del Pobil, A. P., Marta, E., Serna, M. A. (1995a). Dealing with uncertainty in fine motion: a neural approach, in G. F. Forsyth and M. Ali, editors, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 119–126. Gordon and Breach Publishers (Amsterdam).
  - Cervera, E., del Pobil, A. P. (1995c). Multiple self-organizing maps for supervised learning, in J. Mira and F. Sandoval, editors, *From Natural to Artificial Neural Computation*, pages 345–352. Springer Lecture Notes in Computer Science 930 (Berlin).
  - Cervera, E., del Pobil, A. P. (1996). Learning and classification of contact states in robotic assembly tasks, in T. Tanaka, S. Ohsuga and M. Ali, editors, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 725–730, Gordon and Breach Publishers (Amsterdam).
  - Cervera, E., del Pobil, A. P. (1997a). Integration of self-organizing maps and reinforcement learning in robotics, in J. Mira, R. Moreno-Diaz and J. Cabestany, editors, *Biological and Artificial Computation: From Neuroscience to Technology*, pages 1344–1354. Springer Lecture Notes in Computer Science 1240 (Berlin).

- International conference proceedings

  - Cervera, E., del Pobil, A. P., Marta, E., Serna, M. A. (1995b). A sensor-based approach for motion in contact in task planning, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, **2**, pages 468–473, Pittsburgh, USA.
  - Cervera, E., del Pobil, A. P. (1995a). Geometric reasoning for fine motion planning. In *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, pages 154–159, Pittsburgh, USA.

– Cervera, E., del Pobil, A. P. (1995b). A hybrid qualitative connectionist approach to robotic spatial planning. In *Proceedings of the IJCAI Workshop on Spatial and Temporal Reasoning, International Joint Conference on Artificial intelligence (IJCAI'95)*, pages 37–46, Montreal, Canada.

– Cervera, E., del Pobil, A. P. (1997d). Programming and learning in real-world manipulation tasks, in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 471–476, Grenoble, France.

– Cervera, E., del Pobil, A. P. (1997b). Learning strategies for sensor-based manipulation tasks. In *Proceedings of the IEEE International Conference on Computational Intelligence in Robotics and Automation (CIRA'97)*, pages 54–59, Monterey, USA.

6.3 Future work

The integration of learning methods is a challenging field of the AI community. This thesis has presented some successful applications of combinations of SOMs, Q-learning and recurrent networks, with emphasis on a modular design of an autonomous agent architecture. A direct extension is the implementation of Q-learning with recurrent networks in the real robot, in order to test if it improves the learned skills.

Quite a few theoretical questions remain open and deserve future research, leading to fundamental issues: what is learnable, and how can it be learntΓ The *learnable thing* has been decomposed in inputs, features, states, past history, but automatic methods for the selection and the appropriateness of such concepts need to be clarified.

The *empirical bias* of this thesis should be considered, as mentioned before, as guided by the *engineering goal* of artificial intelligence: starting from a real problem, we have used the existing theory and developed new ideas to find a feasible solution. Simulations were intermediate steps towards the real implementation. Collateral contributions to the theory of neural networks and reinforcement learning have emerged from this research. We have tried to avoid the (still very common, sadly) approach of developing a new awesome theory which is only applied to *toy* problems.

In addition to theoretical issues, applications have just begun to be explored. The emphasis on solving real tasks which has guided this thesis should not remain limited to the laboratory environment (which is not as *real* as the real world). Our robot is like a little child, learning while playing

with a piece set with different shapes, and putting each one in the appropriately shaped hole. But that is the means, not the target. When he/she achieves good skills, the next step will be directing his/her attention to the countless objects of everyday use at home, and the unlimited ways of combining them. The use of robots at home, helping disabled people, poses the necessity of skillful manipulators. Nearly all the imaginable home tasks involve some kind of fine motion and contacts with objects.

Industrial applications are undoubtly another important field to benefit from skilled manipulation, in terms of production time, robustness, flexibility, and, needless to say, economic costs. To investigate the industrial assembly tasks which the presented architecture is applicable to, is not a scientific curiosity but a compelling must.

Though skillful with force sensing, our robot is completely blind, deaf, and as dumb as the computer which definitely is its *brain*. A challenging extension is the inclusion of other sensing devices in this approach, significantly enlarging the range of solvable tasks. Vision is what immediately comes to the mind. Though very different in nature to tactile sensing, each one has its weaknesses and strengths: a hybrid approach should benefit from the latter and minimize the former. Vision and force sensing are complementary: vision is useful for a first approach to the goal, and force sensing can solve the final small discrepancies.

In dreaming of robotics, one envisions skillful robots which can solve boring everyday tasks, or cumbersome industrial applications, with robustness, flexibility and simplicity. Robots which, with minimal user interaction, can improve their skills, and reuse their knowledge for new tasks. Solving the problems suggested by this dream demands the resources of numerous fields, including Mechanical Engineering, Physics, Computer Science, Mathematics, and Artificial Intelligence. It is the merging of these fields which makes the task of fulfilling the dream enjoyable.

Appendix A

The Zebra ZERO robot

Technical details of the Zebra ZERO robot arm are described in this appendix. A full description is provided in the technical manual ([Int, 1994]).

A.1 Kinematic configuration

The Zebra ZERO manipulator consists of six links connected with joints as shown in Fig. A.1. The base plate, which stays fixed, is sometimes referred to as an additional link, Link 0. Link 1, the rotating base carriage, is connected to Link 0 through Joint 1, the base rotation. Link 2, the upper arm, is connected to link 1 through Joint 2, the shoulder elevation, etc. The wrist mounting flange and everything which is rigidly attached to it is Link 6. These joints and links will be referred to as J1 - J6 and L1 - L6.

The kinematic configuration of the ZERO is similar to that of the PUMA manipulators, but without some of the small offsets. The J1 axis is normal to the base plate. The J2 axis is orthogonal to, and intersects, the J1 axis. The J3 axis is parallel to the J2 axis.

The plane of the arm is defined as the plane in which the J1 axis lies, and to which the J2 axis is normal. Conceptually, this is the plane formed by the upper arm and forearm links.

The J4 axis is orthogonal to, and intersects, the J3 axis. The J4 axis also lies in the plane of the arm. The J5 axis intersects, and is orthogonal to, the J4 axis. The J6 axis intersects, and is orthogonal to, the J5 axis. All three of these axes intersect at a single point called *wrist center*. The wrist center lies in the plane of the arm. The significant differences between the ZERO kinematics and the PUMA kinematics are that the PUMA wrist center does not lie in the plane of the arm, and the PUMA J4 axis does not intersect the J3 axis.

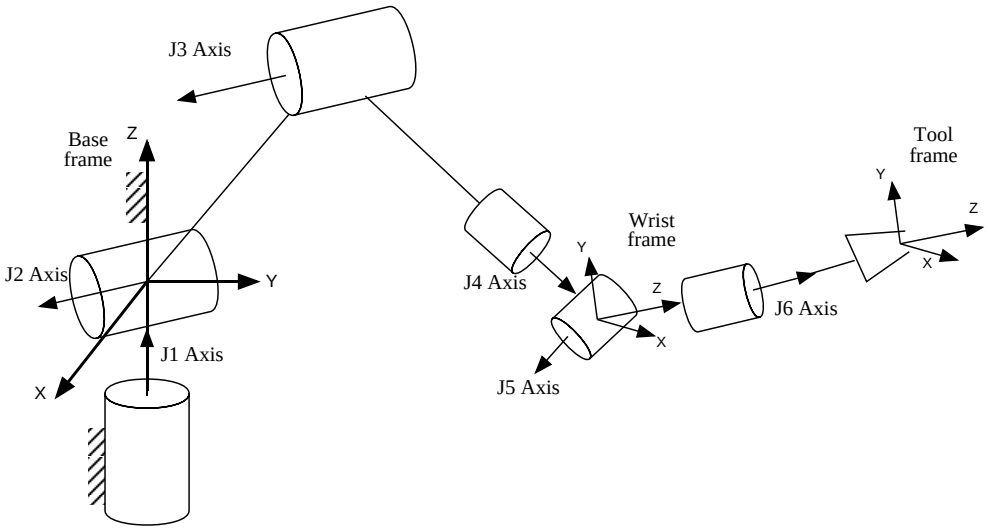


Figure A.1: Zebra ZERO kinematic configuration

Three important Cartesian frames exist: the base frame, the wrist frame, and the tool frame. The base frame is fixed in the base of the arm. The wrist frame is embedded in L6. (The force sensor and the gripper are also part of L6). Its origin is located at the intersection of the wrist axes, it is oriented such that its Z axis points along the J6 axis, and its X axis points normal to the plane of the gripper. The tool frame is a user defined frame which describes the location of the business end of the end-effector relative to the wrist frame. This frame is usually set to correspond to the location of the fingertips or to the location of the point of contact on a part or tool which is being grasped.

A.2 Drive system

The Zebra-ZERO drive system consists of D. C. brush motors driving each joint through a combination of shafts and gears. Each motor has a two stage planetary gear head with reductions 24:1. The motors for driving J1 and J2 are mounted in the rotating base carriage and transmit the power directly through the gears. The motors for J3 and for J4, J5 and J6 are all mounted in the upper arm and act as a counterbalance for the rest of the arm. The power for the wrist joints is transmitted up the arm via shafts and through the elbow joint via idler bevel gears. The wrist uses a concentric shaft differential for driving its three intersecting joints.

Incremental optical encoders are mounted directly on each motor shaft

for position feedback. In general, the motion of any single motor will affect all six joint angles, although the mapping between encoder counts and joint angles is linear.

A.3 Force sensor

The force sensor is a cylindrical unit mounted between the wrist flange and the end-effector. It consists of 3 binding beams, each instrumented with 2 pairs of semiconductor strain gauges, totalling 6 strain measurements. As forces and moments are applied to the end-effector, these beams flex, providing a 6×1 strain measurement vector proportional to the applied 6×1 force/moment vector.

These strain signals are fed into a data acquisition system located at the sensor, which digitizes the signals and sends them serially back to the motor control board. On the card, these signals are reassembled into parallel words and are read by the PC. The strain vector is multiplied by a 6×6 calibration matrix to produce a 6×1 force/moment vector in tool coordinates.

The data acquisition uses a 10 bit A/D converter, providing a force sensor dynamic range of 1000:1. The gain is set for a minimum force reading of about 15 grams.

A.4 Operating modes

A.4.1 Position control

The trajectory of the arm is specified by a series of via points which the arm will pass near or through, and a final goal point where the arm will come to rest. These via points and goal points describe the configuration of the arm in all six degrees of freedom, and can be specified as a 6×1 joint vector, or as a Cartesian frame locating the tool relative to the arm base.

With position control, the motion of the arm is pre-determined at the beginning of the move; that is, it will always try to follow the specified trajectory regardless of the forces applied at the end-effector.

A.4.2 Force threshold mode

When moving the arm under position control, it is possible to set a force threshold, such that if the end-effector feels a force greater than this threshold, the move command will be aborted and motion will stop immediately. The force threshold is specified as a 6×1 vector of three forces and three

torques about the coordinate axes of the tool frame. If the magnitude of any component of the force exceeds the corresponding value in the threshold, the move will abort and the move function will return a value indicating that the force threshold has been exceeded.

To prevent damage from unexpected collisions, a fairly large force threshold, greater than any expected force, should be set. This should be done whenever there is a chance of collision, or whenever the gripper is closed on a part which could be stuck in its fixture. Force thresholds can be used for aligning parts with positional uncertainty by setting a small threshold in the direction along which the end-effector and the other part will first touch. When the end-effector is moved towards the fixed part slowly, the motion will stop when the parts are aligned.

A.4.3 Force control mode

The robot can be placed in a mode where it exerts a specified force on the environment, independent of its position. This force is specified as a 6x1 force/torque vector in tool coordinates. This mode is used whenever the end-effector is completely constrained, as it is when in its homing nest. This mode is also useful for guiding the arm by hand. By commanding the end-effector to exert a zero force, it will comply to forces exerted on the end-effector so as to maintain a zero force.

A.4.4 Stiffness control mode

In most contact situations, the end-effector will be constrained in some directions, but not in others. The stiffness control mode is a means of controlling positions in some directions, and forces in others. In stiffness mode, the user specifies the desired Cartesian stiffness behavior of the end-effector with a 6×1 stiffness vector, representing the translational stiffness of the end-effector along the tool coordinate axes, and the rotational stiffness about these axes. As the end-effector is moved through space, it will deflect from its nominal trajectory in accordance with these stiffness values.

A.5 Homing the robot

When the ZERO robot system is first powered up, it has no idea of where the arm is. To solve this problem, the system should always be started with the arm in known location: in its homing nest. With the arm in the nest, the system has an approximate idea of where the arm is. However, there is

some play between the alignment pins and the nest, and the drive train shafts may be flexed somewhat by friction, so that by reading the motor encoders, the system still won't have an accurate idea of the arm configuration. This problem is solved using the active force sensing and control.

The system first moves the end-effector up and out of the nest, maintaining the same orientation as when in the nest. Once free of the nest, it takes an offset reading to zero the force sensor. The end-effector then moves back into the nest with the aid of a little wobble. The end-effector is next commanded to exert a specific force and moment against the nest. When it reaches this force, the alignment pins will all be pressed firmly against the mating slots, and the arm drive trains will all be flexed by a known amount. At this point, the optical encoder counters are set to zero. The use of force control enables the system to put the arm in a very repeatable configuration.

Appendix B

Software packages

Several software packages, both commercial and public domain, have been used in the implementation of the programs.

B.1 Self-organizing maps

Experiments with SOMs on Chap. 4 were carried out with the public domain software package SOMPAK [Kohonen et al., 1995] which consists of a set of programs written in C for the training and visualization of SOMs. Additional functions for clustering and visualization were written in MATLAB. All the programs run on a Silicon Graphics workstation with the IRIX operating system.

B.2 Recurrent neural networks

The training of Elman networks of Sect. 3.5 was carried out with the Stuttgart Neural Network Simulator [Zell et al., 1995], with additional programs written in MATLAB for the extraction of the automata and the simulations of the agent. Additional processing (including minimization and visualization) of the automata was performed with the public domain FSA Utilities [van Noord, 1995], a set of programs running on SICStus Prolog. All these packages were running on a Silicon Graphics workstation.

B.3 Learning architecture

The learning architecture of Chap. 5 was written in C on a IBM-compatible personal computer with MS-DOS, which controlled the Zebra Zero robot

arm. The program included an implementation of the SOM and Q-learning algorithms.

Bibliography

J. T. Alander, M. Frisk, L. Holmstrom, A. Hamalainen, and J. Tuominen. Process error detection using self-organizing feature maps. Technical report, Rolf Nevanlinna Institute, Helsinki, 1991.

H. Asada. Representation and learning of nonlinear compliance using neural nets. *IEEE Transactions on Robotics and Automation*, 9(6):863–867, 1993.

A. G. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

M. Brady. Artificial intelligence and robotics. *Artificial Intelligence*, 26: 79–121, 1985.

A. J. Briggs. An efficient algorithm for one-step planar compliant motion planning with uncertainty. In *5th ACM Annual Symposium on Computational Geometry*, pages 187–196, 1989.

M. E. Caine, T. Lozano-Pérez, and W. P. Seering. Assembly strategies for chamferless parts. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 472–477, 1989.

T. P. R. Campos. Connectionist modeling for arm kinematics using visual information. *IEEE Transactions on Systems, Man, and Cybernetics*, 26 (1):89–99, 1996.

J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28th IEEE Symposium on Foundations of Computer Science*, pages 49–70, 1987.

R. C. Carrasco and M. L. Forcada. Second-order recurrent neural networks can learn regular grammars from noisy strings. In J. Mira and F. Sandoval, editors, *From Natural to Artificial Neural Computation*, number 930 in Lecture Notes in Computer Science, pages 605–610. Springer, 1995.

M. A. Castaño, E. Vidal, and F. Casacuberta. Finite state automata and connectionist machines: A survey. In J. Mira and F. Sandoval, editors, *From Natural to Artificial Neural Computation*, number 930 in Lecture Notes in Computer Science, pages 433–440. Springer, 1995.

E. Cervera and A. P. del Pobil. Geometric reasoning for fine motion planning. In *Proccedings of the IEEE International Symposium on Assembly and Task Planning*, pages 154–159, 1995a.

E. Cervera and A. P. del Pobil. A hybrid qualitative-connectionist approach to robotic spatial planning. In *Proceedings of the IJCAI Workshop on Spatial and Temporal Reasoning*, pages 37–46, 1995b.

E. Cervera and A. P. del Pobil. Multiple self-organizing maps for supervised learning. In J. Mira and F. Sandoval, editors, *From Natural to Artificial Neural Computation*, number 930 in Lecture Notes in Computer Science, pages 345–352. Springer, 1995c.

E. Cervera and A. P. del Pobil. Learning and classification of contact states in robotic assembly tasks. In T. Tanaka, S. Ohsuga, and M. Ali, editors, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 725–730. Gordon and Breach Publishers, 1996.

E. Cervera and A. P. del Pobil. Integration of self-organizing feature maps and reinforcement learning in robotics. In J. Mira, R. Moreno-Diaz, and J. Cabestany, editors, *Biological and Artificial Computation: From Neuroscience to Technology*, number 1240 in Lecture Notes in Computer Science, pages 1344–1354. Springer, 1997a.

E. Cervera and A. P. del Pobil. Learning strategies for sensor-based manipulation tasks. In *Proceedings of the International Conference on Computational Intelligence in Robotics and Automation (CIRA)*, pages 54–59, 1997b.

E. Cervera and A. P. del Pobil. Multiple self-organizing maps: A hybrid learning approach. *Neurocomputing*, 1997c. In press.

E. Cervera and A. P. del Pobil. Programming and learning in real-world robotic tasks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 471–476, 1997d.

E. Cervera, A. P. del Pobil, E. Marta, and M. A. Serna. Dealing with uncertainty in fine motion: a neural approach. In G. F. Forsyth and M. Ali, editors, *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 119–126. Gordon and Breach Publishers, 1995a.

E. Cervera, A. P. del Pobil, E. Marta, and M. A. Serna. A sensor-based approach for motion in contact in task planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 468–473, 1995b.

E. Cervera, A. P. del Pobil, E. Marta, and M. A. Serna. Perception-based learning for motion in contact in task planning. *Journal of Intelligent and Robotic Systems*, 17:283–308, 1996.

A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning reliable manipulation strategies without initial physical model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1224–1230, 1990.

A. Cleeremans, D. Servan-Schreiber, and J. L. McClelland. Finite state automata and simple recurrent networks. *Neural Computation*, 1:372–381, 1989.

C. Darken and J. Moody. Note on learning rate schedule for stochastic optimization. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 832–838. Morgan Kaufmann, 1991.

A. P. del Pobil and M. A. Serna. *Spatial Representation and Motion Planning.* Number 1014 in Lecture Notes in Computer Science. Springer Verlag, 1995.

D. DeMers and K. Kreutz-Delgado. Canonical parameterization of excess motor degrees of freedom with self-organizing maps. *IEEE Transactions on Neural Networks*, 7(1):43–55, 1996.

R. J. Desai and R. A. Volz. Identification and verification of termination conditions in fine motion in presence of sensor errors and geometric uncertainties. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 800–807, 1989.

P. A. Devijver and J. Kittler. *Pattern Recognition: a Statistical Approach.* Prentice Hall, 1982.

B. R. Donald. *Error Detection and Recovery in Robotics.* Springer Verlag, 1989.

S. Dutré, H. Bruyninckx, and J. de Schutter. Contact identification and monitoring based on energy. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

M. A. Erdmann. Using backprojections for fine motion planning with uncertainty. *International Journal of Robotics Research*, 5(1):19–45, 1986.

K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Computer Science and Scientific Computing. Academic Press, 2nd edition, 1990.

C. L. Giles, C. B. Miller, D. Chen, G. Z. Sun, H. H. Chen, and Y. C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4:393–405, 1992.

R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

S. Gottschlich, C. Ramos, and D. Lyons. Assembly and task planning: A taxonomy. *IEEE Robotics and Automation Magazine*, pages 4–12, 1994.

V. Gullapalli, J. A. Franklin, and H. Benbrahim. Acquiring robot skills via reinforcement learning. *IEEE Control Systems*, 14(1):13–24, 1994.

V. Gullapalli, R. A. Grupen, and A. G. Barto. Learning reactive admittance control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1475–1480, 1992.

A. S. Gusev. Automatic assembly of cylindrically shaped parts. *Russian Engineering Journal*, 49(11):53, 1969.

M. Hernández-Pajares. How tracer object can improve competitive learning algorithms in astronomy. *Vistas in Astronomy*, 38:317–330, 1994.

J. A. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, 1991.

S. Hirai and K. Iwata. Recognition of contact state based on geometric model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1507–1512, 1992.

J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Series in Computer Science. Addison-Wesley, 1979.

G. E. Hovland and B. J. McCarragher. Frequency-domain force measurements for discrete event contact recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1166–1171, 1996.

G. E. Hovland, P. Sikka, and B. J. McCarragher. Skill acquisition from human demonstration using a Hidden Markov Model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2706–2711, 1996.

Integrated Motions Inc, 758 Gilman Street, Berkeley, California 94710. *Zebra ZERO Force Control Robot: User Guide*, 1994.

J. Jennings, B. R. Donald, and D. Campbell. Towards experimental verification of an automated compliant motion planner based on a geometric theory of error detection and recovery. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 632–637, 1989.

M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *8th Annual Conference of the Cognitive Science Society*, pages 531–546, 1986.

L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

M. Kaiser and R. Dillman. Hierarchical learning of efficient skill application for autonomous robots. In *International Symposium on Intelligent Robotic Systems*, 1995.

M. Kaiser and R. Dillman. Building elementary robot skills from human demonstration. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2700–2705, 1996.

K. Kasslin, J. Kangas, and O. Simula. Process state monitoring using self-organizing maps. In I. Aleksander and J. Taylor, editors, *Artificial Neural Networks*, volume 2. Elsevier Science Publishers, 1992.

S. Keerthi and B. Ravindran. A tutorial survey of reinforcement learning. In E. Fiesler and R. Beale, editors, *Handbook of Neural Computation*. Oxford University Press, 1996.

T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

T. Kohonen. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer, 1995.

T. Kohonen, G. Barna, and R. Chrisley. Statistical pattern recognition with neural networks. In *IEEE International Conference on Neural Networks*, volume 1, pages 61–68, 1988.

T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen. *SOMPAK: The Self-Organizing Map Program Package*. Helsinki University of Technology, 1995.

M. A. Kraaijveld, J. Mao, and A. K. Jain. A non-linear projection method based on Kohonen's topology preserving maps. In *11th International Conference on Pattern Recognition*, pages 41–45, 1992.

S. C. Kremer. On the computational power of Elman-style recurrent networks. *IEEE Transactions on Neural Networks*, 6(4):1000–1004, 1995.

B. J. A. Kröse and Marc Eecen. A self-organizing representation of sensor space for mobile robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1994.

K. Kurimo. Hybrid training method for tied mixture density Hidden Markov Models using LVQ and Viterbi estimation. In *IEEE Workshop on Neural Networks for Signal Processing*, pages 362–371, 1994.

N. M. Laktionev and G. Y. Andreev. Automatic assembly of parts. *Russian Engineering Journal*, 46(8):40, 1966.

J. C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, 52(1):1–47, 1991.

C. Laugier. Planning fine motion strategies by reasoning in the contact space. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 653–659, 1989.

L. J. Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie Mellon University, 1993.

T. Lozano-Pérez. Spatial planning: a configuration space approach. *IEEE Transacions on Computing*, 32(2):108–120, 1983.

T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

P. Manolios and R. Fanelli. First-order recurrent neural networks and finite state automata. Technical report, Brooklyn College of the City University of New York, 1993.

B. Martin del Brio and C. Serrano-Cinca. Self-organizing neural networks for the analysis and representation of data: Some financial cases. *Neural Computing and Applications*, 1:193–206, 1993.

M. T. Mason. Compliance and force control for computer controlled manipulators. *IEEE Transactions on Systems, Man and Cybernetics*, 11:418–432, 1981.

M. T. Mason. Compliant motion. In M. Brady, J. M. Hollerbach, T. L. Johnson, T. Lozano-Pérez, and M. T. Mason, editors, *Robot Motion: Planning and Control*, pages 305–322. The MIT Press, 1982.

B. J. McCarragher. Force sensing from human demonstration using a hybrid dynamical model and qualitative reasoning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 557–563, 1994.

B. J. McCarragher. The unsupervised learning of assembly using discrete event control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1172–1177, 1996.

B. J. McCarragher and H. Asada. A discrete event controller using Petri nets applied to assembly. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2087–2094, 1992.

B. J. McCarragher and H. Asada. A discrete event approach to the control of robotic assebly tasks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 331–336, 1993a.

B. J. McCarragher and H. Asada. Qualitative template matching using dynamic process models for state transition recognition of robotic assembly. *ASME Journal of Dynamic Systems, Measurement and Control*, 114(2), 1993b.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

J. R. Millán and C. Torras. A reinforcement connectionist approach to robot path-finding in no-maze-like environments. *Machine Learning*, 8:363–395, 1992.

N. Mimura and Y. Fubahashi. Parameter identification of contact conditions by active force sensing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2645–2650, 1994.

M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, New York, 1967.

J. Mira, R. Moreno-Diaz, and J. Cabestany, editors. *Biological and Artificial Computation: From Neuroscience to Technology*. Number 1240 in Lecture Notes in Computer Science. Springer, 1997.

J. Mira and F. Sandoval, editors. *From Natural to Artificial Neural Computation*. Number 930 in Lecture Notes in Computer Science. Springer, 1995.

T. M. Mitchell and S. Thrun. Explanation-based neural network learning for robot control. In S. J. Hanson, J. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 287–294. Morgan Kaufmann, 1993.

J. D. Morrow and P. K. Khosla. Sensorimotor primitives for robotic assembly skills. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1894–1899, 1995.

M. C. Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381, 1989.

B. K. Natarajan. The complexity of fine motion planning. *International Journal of Robotics Research*, 7(2):36–42, 1988.

J. Naylor, A. Higgins, and K. P. Li. Speaker recognition using Kohonen's self-organizing feature map algorithm. *Neural Networks*, 1:311, 1988.

D. Niebur and A. J. Germond. Unsupervised neural net classification of power system static security states. *Electrical Power and Energy Systems*, 14:233–242, 1992.

M. Nuttin, J. Rosell, R. Suárez, H. van Brussel, L. Basáñez, and J. Hao. Learning approaches to contact estimation in assembly tasks with robots. In *3rd European Workshop on Learning Robots*, 1995.

M. Nuttin, H. van Brussel, C. Baroglio, and R. Piola. Fuzzy controller synthesis in robotic assembly: Procedure and experiments. In *3rd IEEE International Conference on Fuzzy Systems*, pages 1217–1223, 1994.

W. Paetsch and G. von Wichert. Solving insertion tasks with a multifingered gripper by fumbling. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 173–179, 1993.

B. D. Ripley. Neural networks and related methods for classification. *Journal of the Royal Statistical Society B*, 56:509–456, 1994.

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

A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University, 1989.

S. M. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, 1983.

V. Ruiz de Angulo and C. Torras. Self-calibration of a space robot. *IEEE Transactions on Neural Networks*, 8(4):951–963, 1997.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362, Cambridge MA, 1986. MIT Press.

P. Sikka and B. J. McCarragher. Monitoring contact using clustering and discriminant functions. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1351–1356, 1996.

S. N. Simunovic. Force information in assembly processes. In *5th International Symposium on Industrial Robots*, pages 415–431, Chicago, 1975.

M. Skubic and R. A. Volz. Identifying contact formations from sensory patterns and its applicability to robot programming by demonstration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 2, pages 458–464, 1996.

M. Spreng. A probabilistic method to analyze ambiguous contact situations. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 543–548, 1993.

W. S. Stornetta, T. Hogg, and B. A. Huberman. A dynamical approach to temporal pattern procesing. In *Neural Information Processing Systems*, pages 750–759, Denver, 1988.

R. Suárez and L. Basáñez. Assembly with robots in presence of uncertainty. In *International Robots and Vision Conference*, pages 1–15, 1991.

R. Suárez and L. Basáñez. Illustrating an automatic planner for robotic assembly task. In *23rd International Symposium on Industrial Robots*, pages 645–651, Barcelona (Spain), 1992.

R. Suárez, L. Basáñez, and J. Rosell. Assembly contact force domains in the presence of uncertainty. In *4th IFAC Symposium on Robot Control*, pages 653–659, Capri (Italy), 1994.

R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, 1984.

R. S. Sutton. Integrated modeling and control based on reinforcement learning and dynamic programming. In R. P. Lippman, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, pages 471–478. Morgan Kaufmann, 1991.

R. S. Sutton, editor. *Reinforcement Learning.* Kluwer Academic Publishers, 1992.

S. B. Thrun. The role of exploration in learning control. In D. A. White and D. A. Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches.* Van Nostrand Reinhold, 1992.

S. B. Thrun. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach.* Kluwer Academic Publishers, 1996.

C. Torras. From geometric motion planning to neural motor control in robotics. *AI Communications*, 6(1):3–17, 1993.

V. Tryba and K. Goser. Self-organizing feature maps for process control in chemistry. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 847–852. Elsevier Science Publishers, 1991.

A. Ultsch. Self-organizing feature maps for monitoring and knowledge acquisition of a chemical process. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 864–867, 1993.

G. van Noord. *FSA Utilities Manual*. University of Groningen, 1995.

M. Vapola, O. Simula, T. Kohonen, and P. Merilainen. Representation and identification of fault conditions of an anaesthesia system by means of the self-organizing map. In M. Marinaro and P. G. Morasso, editors, *Proceedings of the International Conference on Artificial Neural Networks*, pages 350–353, 1994.

S. Vougioukas and S. Gottschlich. Automatic synthesis and verification of compliance mappings. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 491–496, 1993.

J. A. Walter and K. J. Schulten. Implementation of self-organizing neural networks for visuo-motor control of an industrial robot. *IEEE Transactions on Neural Networks*, 4(1):86–95, 1993.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

D. E. Whitney. Quasi-static assembly of compliantly supported rigid parts. *ASME Journal of Dynamic Systems, Measurement and Control*, 104:65–77, 1982.

D. E. Whitney. Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, 6(2):3–14, 1987.

P. M. Will and D. D. Grossman. An experimental system for computer controlled mechanical assembly. *IEEE Transactions on Computers*, 24(9): 879–888, 1975.

P. H. Winston. *Artificial Intelligence*. Addison-Wesley, 3rd edition, 1992.

J. Xiao. Automatic determination of topological contacs in the presence of sensing uncertainties. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 65–70, 1993.

J. Xiao and R. A. Volz. On replanning for assembly tasks using robots in the presence of uncertainties. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 638–645, 1989.

J. Xiao and L. Zhang. Toward obtaining all possible contacts—growing a polyhedron by its location uncertainty. *IEEE Transactions on Robotics and Automation*, 12(4):553–565, 1996.

A. Zell et al. *SNNS: Stuttgart Neural Network Simulator, User Manual, Version 4.1.* Institute for Parallel and Distributed High Performance Systems, 1995.