Example Generation and Processing in Robot Programming By Demonstration

Luis Seabra Lopes and Luis Manuel Camarinha-Matos
Departamento de Engenharia Electrotécnica
Universidade Nova de Lisboa
Quinta da Torre
2825 Monte da Caparica
PORTUGAL

Abstract: An architecture for supervision and programming by demonstration in robotised assembly is presented. This architecture provides, at different levels of abstraction, functions for dispatching actions, monitoring their execution, and diagnosing and recovering from failures. Through the use of machine learning techniques, the supervision architecture is given capabilities for improving its performance over time. Emphasis in this paper is on generating accurate classification knowledge for diagnosis. Feature construction, example interpolation and hierarchical classification are the investigated alternatives for improving accuracy. The SKIL hierarchical inductive learning algorithm is briefly presented.

Keywords: Robotised Assembly, Failure Diagnosis, Programming by Demonstration, Feature Construction, Example Interpolation, Multi-Level Induction

1. Introduction

The ability to produce highly customized products, in order to satisfy market niches, is a competitive advantage that companies are trying to take, in today's globalized economy. This requires the introduction of new features in automation systems — flexibility, adaptability, versatility — relaxing cell structuring constraints and leading to the concept of Flexible Manufacturing Systems (FMS).

To accommodate the wide range of product-processing techniques in an FMS more versatile machines must be considered [1,18]: easily integratable CNC machines, multi-operation devices, robots with multiple tools / end-effectors, modular fixtures and feeders. A rich sensorial environment and advanced communication infrastructures (and related protocols: MAP/MMS, fieldbus, ...) are other important hardware requirements to build up an FMS.

The efficiency and economical success of flexible manufacturing systems depend, however, on the capacity to handle unforeseeable events, which occur in a great number, due to the reduction in system structuring constraints. The complexity of flexible manufacturing processes makes this task difficult to perform by humans. Therefore, in manufacturing systems, flexibility and autonomy are tightly related concepts. The introduction of "intelligent" functionalities, at programming level, is vital to achieving flexibility and autonomy: interactive programming / planning; sensorial perception and status identification; on-line decision-making capabilities, to cope with the non-deterministic behaviour of less structured cells; information integration, etc.

With these concerns in mind, an architecture for evolutive supervision of robotised assembly tasks, that includes, at different levels of abstraction, functions for dispatching actions, for monitoring their execution, and diagnosing and recovering from failures, has been proposed [1]. In this architecture, particular attention was given to the acquisition of knowledge about the tasks and the environment to support monitoring, to diagnosis and error recovery. Modeling execution failures through taxonomies and causal networks plays a central role in diagnosis and recovery.

A generic training methodology was devised [18], in which a human tutor could provide the system with a priori knowledge, with examples of desired behaviour and examples of concepts to be learned (programming by demonstration). Training is considered both at initial system programming and when unpredicted situations turn up. Based on the demonstrated examples and using machine learning techniques, the system will incrementally build the needed models. An overview of the approach is included below.

This paper concentrates on learning of diagnosis knowledge, with emphasis on example generation, numeric to symbolic conversion and feature construction and selection. An inductive learning algorithm, SKIL, previously proposed by the authors [17], is used in the experiments and compared, in terms of performance, to a Nearest Neighbour classifier [2] and to a supervised clustering algorithm, Q* [15]. One of the main problems in previous research was the poor classification accuracy. Progress in this respect is reported. Several alternative approaches, investigated in the context of the ESPRIT project B-LEARN II, are described in detail.

2. Robot Programming By Demonstration

A crucial point in flexible manufacturing systems is the acquisition of knowledge to support the supervision functions. In real execution, a feature extraction function is permanently acquiring monitoring features from the raw sensor data (namely from force & torque data and from discrete sensor data). These features are used to guide the control function of the operation being executed as well as to detect deviations between the actual behaviour and the nominal operation behaviour. For example, let us consider that, during the execution of a `Transfer` operation, in which the robot carries a part to be assembled, an object, unexpectedly originating in the environment, collides with the gripper causing the part to move without falling. The first diagram, included in Figure 1, shows the perceived sensor data during actual execution. The second diagram shows a qualitative model of the operation. The third diagram shows a qualitative interpretation of the raw sensor data in terms of the features used in the operation model. Since a deviation is detected, the diagnosis function is called to verify if an execution failure occurred and, in that case, to determine a failure classification and explanation. For this function, additional features must be extracted. Diagnosis is a decision procedure that needs a more sophisticated model of the task, the system and the environment. The final step, based on the failure characterization, is recovery planning.

The problem of building the knowledge base, and in particular the models that the monitoring, diagnosis and recovery functions need, is not easy to solve. Even the best domain expert will have difficulty in specifying the necessary mappings between the available sensors on one side and the monitoring conditions, failure classifications, failure explanations and recovery strategies on the other side. Also, a few less common errors will be forgotten. Known prototype systems show limited domain knowledge, as they are intended mainly for exemplification and not to be used as robust solutions in the real world.

In a very initial phase, the models of the elementary operations of the system must be built, considering both sensing and action aspects. Some operation models can be easily hand-coded. For instance, part feeding or robot motion along a given path are operations for which the control function is relatively easy to specify. Ensuring the nominal conditions for these operations is also simple to do. For instance, a few binary sensors are enough to identify the status of a part feeder. To hand-code an evaluation function based on these sensors is a trivial task.

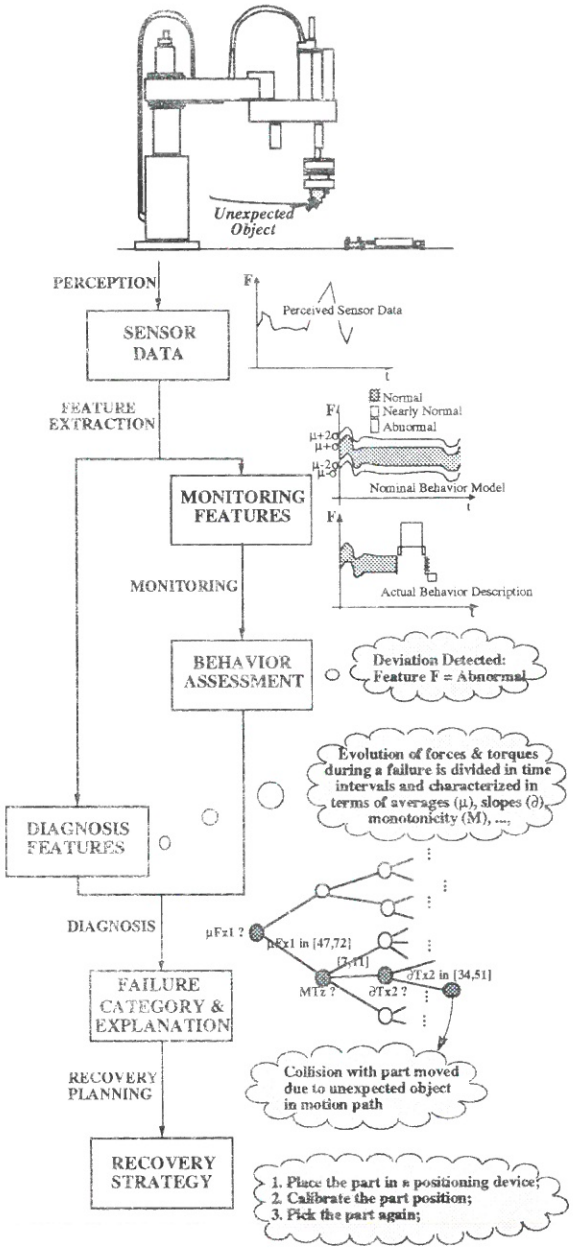


Figure 1. Example of the Error Detection and Recovery Cycle

For complex operations, that do not have a well-defined model, the control functions are very costly to program in the classical way. An example is compliant motion in robotized assembly: the variations in size, weight, friction coefficient, etc., of the parts involved are so wide that a clear generic model of the needed compliance does not exist. The same way, evaluating the behaviour of the system during execution of a given operation can be much more

complex than assessing the status of a feeder based on binary sensors.

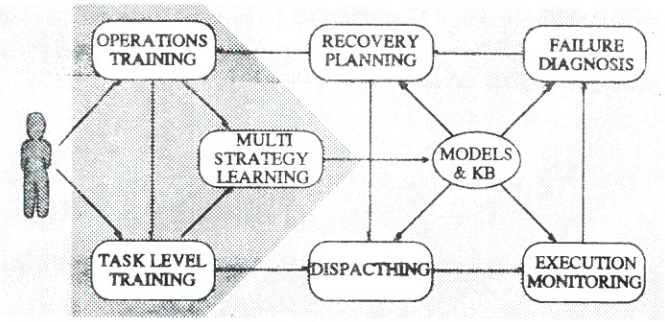


Figure 2.Training and Supervision

The Programming by Human Demonstration paradigm (in this case RPD — *Robot Programming by Demonstration* [5, 6, 8, 9, 18]) seems indicated to overcome such type of difficulties. According to this paradigm, complex systems are programmed by showing particular examples of their desired behaviour. Usually, emphasis is put on robots learning from their own perception of humans performing certain tasks. In our approach, any interaction with the human, that enables the robotic system to perform better in the future, is also included in the programming by demonstration framework, and is referred to as a *training action*. Functions for training and learning are included in the supervisor architecture (Figure 2).

An adequate user interface facilitates transfer of the human's knowledge to the system. This knowledge can be coherent and complete or can be empirical, based on real examples of complex situations which have no known model. For instance, to teach the robot how to insert a peg into a hole (a compliant operation), the tutor can guide the robot arm to reach that goal, while collecting both force & torque data and applied velocities. Then, the best tutoring experiments (according to some criteria, for instance the minimization of insertion time) are selected to serve as examples that a neural network or a fuzzy controller uses to learn the control function of the operation (application of learning in compliant motion is being investigated by our B-LEARN partners [12]).

In what concerns the definition and characterization of elementary operations, our research focus has been centered in execution evaluation, including monitoring (detection of failures) and diagnosis (classification and explanation). For instance, the knowledge required for monitoring of motion operations was obtained by training in the following way: traces of all testable sensors were collected during several runs of each operation; for each continuous feature, the typical behavior of the attribute during execution of the operation was calculated as being the region between the average minus standard deviation behaviour and the average plus standard deviation behaviour.

The system is also trained to identify/classify execution failures. The result is the creation and refinement of a qualitative failure model, composed of failure descriptions and taxonomic and causal relations between them. For training, several external exceptions are manually provoked and the effects as well as the corresponding traces of sensor values are recorded. Examples of provoked external exceptions are: unexpected objects on robot arm motion path; misplaced or missing parts; defective parts or assemblies; misplaced or missing tools. For some of the resulting execution failures, it was easy to hand-code classification rules. For others, classification knowledge is generated by inductive algorithms.

Finally, recovery strategies are programmed for the most common errors. A good user interface should facilitate also the specification of recovery strategies. Feasible possibilities, that we have not considered, are graphical simulation and virtual reality. In the current prototype, recovery strategies are entered in textual form. We are looking at early planning systems as STRIPS/PLANNEX [3], HACKER [19], and also at case-based reasoning and explanation-based generalization techniques in order to incorporate, into the supervisor, learning capabilities at the planning level [18].

In any case, the main idea is that programming the supervision system includes both traditional programming (implanting in the system monitoring, diagnosis and recovery rules, known *a priori*) and programming by demonstration (examplar-based approach).

It is convenient to distinguish between training of elementary operations and specific task-level training (Figure 2). In the first case, the implanted knowledge and the provided examples are concerned with characterizing the basic primitives of the system (elementary operations, those used in task planning) and their related skills. Only after this phase, the system should start training and then executing specific tasks.

In real execution, when a failure is identified, and there is no known recovery procedure, recovery planning is attempted. If a plan is found and, when applied, brings the system back to nominal

state, this plan will be generalized and stored for further use. If it is not possible to recover automatically, help from a human expert is requested. Eventually, the solution provided by the human is also generalized and stored. This kind of incremental programming and generalization may also be viewed as training. At the level of elementary operations, execution evaluation contributes to the tuning of the related skills [5,12].

This approach allows for progressive independence of the human operator, who will not have to be permanently supervising the assembly process. On the other hand, as the system relies more on continuous self-evaluation of its behaviour than on very accurate models, the requirements on operator specialization are relaxed. The main limitation of the proposed approach is the cost of training in the initial phase: generating a sufficient number of examples of each learning concept, mainly when physical devices are involved, is a **tedious task**.

Training and execution make up a long-term learning cycle along which the models of operations, manipulated objects and the environment are built and refined. Experience gained in making a certain product is useful for making other products. In this way, training a specific task is simplified, especially from the humans' perspective. Long-term learning, however, poses many problems concerning knowledge representation, an issue with great impact on reasoning processes.

3. Learning in Diagnosis

In plan execution, the diagnosis function is called when the monitoring function detects off-nominal feedback in the system. Diagnosis can be decomposed into four steps: failure confirmation; failure classification; cause explanation; and status identification. In abstract, the first two steps are classification problems while the remaining two are explanation problems.

3.1 Domain Knowledge

In [1] it was proposed to divide errors into three main families: system faults, external exceptions and execution failures. *Execution failures* are deviations of the state of the world from the expected state detected during the execution of actions. *External exceptions* are abnormal occurrences in the physical environment that may cause execution failures. *System faults* are abnormal occurrences in the assembly resources hardware and software or in the communication systems.

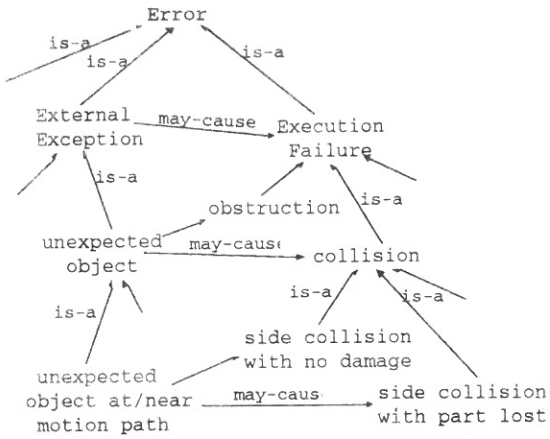


Figure 3. Example of Causal Links at Different Levels of the Error Taxonomy

Depending on the available sensorial information, a more or less detailed classification and explanation of the detected execution failure may be obtained. Therefore, the model of errors should be a taxonomy. At each level of this taxonomy, cause-effect relations between different types of errors should be added. Typically, execution failures are caused by system faults, external exceptions or other past execution failures, although, in general, errors of the three kinds may cause each other. Determining explanations for detected execution failures can become very complex when errors propagate. The proposed approach to modelling errors in terms of taxonomic and causal links aims at handling this complexity (Figure 3).

Training the system to understand the meaning of sensor values and learning a qualitative and hierarchical model of the behaviour of each operation is a crucial step in diagnosis. Programming such model by hand would be nearly impossible. Since the human defines the "words" (attribute names and values) used in the model, the human is capable of understanding a more or less detailed description that the model provides for each situation. It is then easier to hand-code explanations for the situations described in the model. The explanation that must be obtained for the given execution failure includes not only the ultimate cause (an external exception or system fault), but also the determination of the new state of the system [18].

3.2 Learning and Classification

As emphasized above, the difficulty in hand-coding the models that the supervision functions need, raises the question of how to build such models automatically. The classification phase of the diagnosis task can be performed based on knowledge generated by induction.

In the experiments described below, three learning algorithms will be used. The first one is a Nearest Neighbor classifier [2]. This type of classifiers do not perform any data compression on the provided examples. The available classified examples are directly used to classify new examples. The class assigned by a k-Nearest-Neighbor classifier (in short k-NN) to a new sample is the most frequent class in those k examples that, according to a distance metric (typically the Euclidean norm), are closer to this new sample. Since, in most cases, it is difficult to define a distance metrics for qualitative features, k-NN classifiers are more suited for continuous domains. Error estimation based on k-NN classification is usually a good reference for assessing the performance of other algorithms. The absence of data compression is its main limitation. Below, the basic form of the Nearest Neighbor classifier (the 1-NN) will be used.

The second algorithm used is Q*, limited to continuous domains, that performs supervised clustering [15]. It learns a set of *prototypes* (the same as *codebook vectors* in LVQ [7]) of the classes present in the provided training set. The algorithm is self-organizing, since prototypes are dynamically created and modified during the iterative learning procedure. When some training example cannot be classified correctly by the existing set of prototypes, an additional prototype is created. Q* nicely handles multimodal class distributions. To classify unknown samples, after learning, a 1-Nearest-Neighbor strategy is followed on the generated set of prototypes.

The k-Nearest-Neighbor classifier and the Q* algorithm are available in the TOOLDIAG package [14].

Both k-NN and Q* are only able to learn uni-dimensional concept descriptions: the resulting knowledge is only able to assign classes to objects from a given domain. The same can be said about other systems that we have used, previously, in the assembly domain [1,17]. For instance, these algorithms and systems cannot handle the problem of discriminating collisions from obstructions and normal situations, handling simultaneously the discrimination between different types of collisions.

Having as a motivation the automatic construction of the models required for the assembly supervisor, the idea of generating a concept hierarchy has become attractive. The problem of learning at multiple levels of abstraction has not been adequately considered yet in the literature. In some approaches, a fixed decomposition of concepts is used, and learning is applied at each level [11]. However this is not flexible enough. Fixed decompositions have also been used for feature values [11, 16]. A new algorithm, SKIL (structured knowledge generated by inductive learning), was developed to perform this task [17].

The concepts in the hierarchy learned by SKIL are characterized by a set of symbolic classification attributes. At the lower levels of the hierarchy, concepts are described in more detail, i.e. more attribute values are specified. Moreover, in detailing or refining a concept, in which attributes take certain values, it can make sense to calculate other attributes. Therefore, the user may provide a set of attribute enabling statements of the form (A_i, A_{ij}, EA_{ij}), meaning that when the value of A_i is determined to be A_{ij}, then attributes in EA_{ij} should be included in the set of attributes to consider in the continuation of the induction process. For example, when learning the behaviour of a `Transfer` operation, if a collision is found, it can make sense to determine some characteristics of the colliding object, like size, hardness and weight. This could be expressed by the following attribute enabling triple:

(failure_type, collision, {obj_size, obj_hardness, obj_weight})

The attribute values of the concepts in the hierarchy are determined inductively based on training data specified in terms of a set of discrimination features, which can be numerical or symbolic. Each example in the training set is composed of a list of attribute-value pairs followed by a vector of feature values.

The algorithm is a recursive procedure that takes as parameters a list of examples, a list of classification attributes, a list of attribute enabling triples and a list of features. At each stage of the induction, the main goal is to determine the values of as many classification attributes as possible. For each attribute, the discrimination power of features is evaluated, using an entropy measure [13]. For continuous features, segmentation of feature values is done at each decision node, in a way that maximizes its discrimination power [17]. The feature that, for some attribute, gives the lowest entropy is selected to be test feature. Expansion stops when

the values of all attributes have been determined or when it is not possible to extract more information from the data.

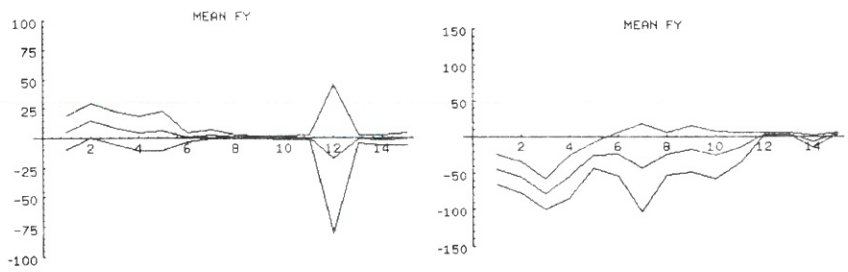
The basic knowledge transmutation used by SKIL is, therefore, an empirical inductive generalisation (see the Inferential Theory of Learning [10]), only that at multiple levels of abstraction. The generated knowledge structure is a hierarchy of anonymous concepts, each of them defined by the combination of several attribute-value pairs. The number of specified attributes and values defines the abstraction level (of course, as a special case, SKIL can also work as any traditional classification algorithm, i.e. with only one classification attribute). The formation of these concepts, guided by the attribute enabling triples, highly depends on the training data. The hierarchy is, simultaneously, a decision tree that can be used to recognize instances of the concepts.

4. Experimentation

From the supervision point of view, the results obtained in previous research, using different learning systems, were not satisfactory, due to the low classification accuracies on unseen examples. Accuracy did not go beyond 90%, and the most typical results were between 50% and 70%.

In robotics applications, information about the status of the system must often be obtained from complex sensors that provide numerical data difficult to analyse. Some failures can easily be identified by simple discrete sensors. For instance, in case the wrong tool is attached to the robot, this can be detected by a binary sensor. Such knowledge can easily be coded by hand as rules. However, how to characterize a situation in which the force profile in the robot wrist is not normal? Different external exceptions can occur causing execution failures that manifest through abnormal force and torque profiles. These profiles, although sometimes recognizable by the human, are difficult to model analytically. Therefore, it is desirable that the system learns to "look" at the force profiles in order to identify different situations.

An obvious explanation for the poor accuracy results obtained in previous experiments was that the relevant features in each situation were not directly "visible" in the sensor data. It was then decided to perform exaustive experiments with different *feature construction* strategies on four learning problems, related to a «pick and place» macro-operation. Other directions for improving accuracy, that were explored, are *multi-level induction* and *example interpolation*. This latter method did provide surprisingly good results.



a) Collision in part & part lost

b) Front collision and part moved

Figure 4. Examples of Typical Force Behaviour During Two Different Types of Failures in *Approach-Ungrasp*

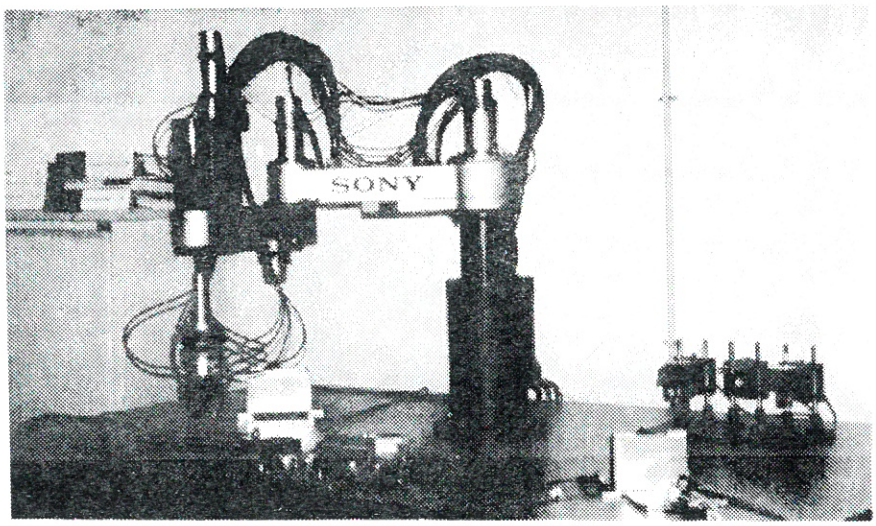


Figure 5. The Experimental Scenario

4.1 Selected Test Problems

Four learning problems were selected for the experiments. In each of them, the goal is to learn to recognize some relevant aspects of an execution failure, by maximizing the accuracy of the learned knowledge. Each collected example is composed of traces of the forces and torques in 3D, in a time interval surrounding the failure. The length of each trace is of 15 measurements, therefore each example is characterized by 90 raw variables. The learning problems are:

LP-1: failures in approach-grasp — four classes of system behaviour, that will be learned, are considered: *normal*, *collision*, *front collision* and *obstruction*; 88 examples were collected.

LP-2: failures in transfer of part — five classes of system behaviour are considered: *normal*, *front collision*, *back collision*, *collision to the right* and *collision to the left*; 47 examples were collected.

LP-3: failures in approach-ungrasp — the classes of failures to be learned are: *normal*, *collision* and *obstruction*; 117 examples were collected.

LP-4: failures in motion with part — five classes of system behaviour are considered: *normal*, *bottom collision*, *bottom obstruction*, *collision in part* and *collision in tool*; 164 examples were collected.

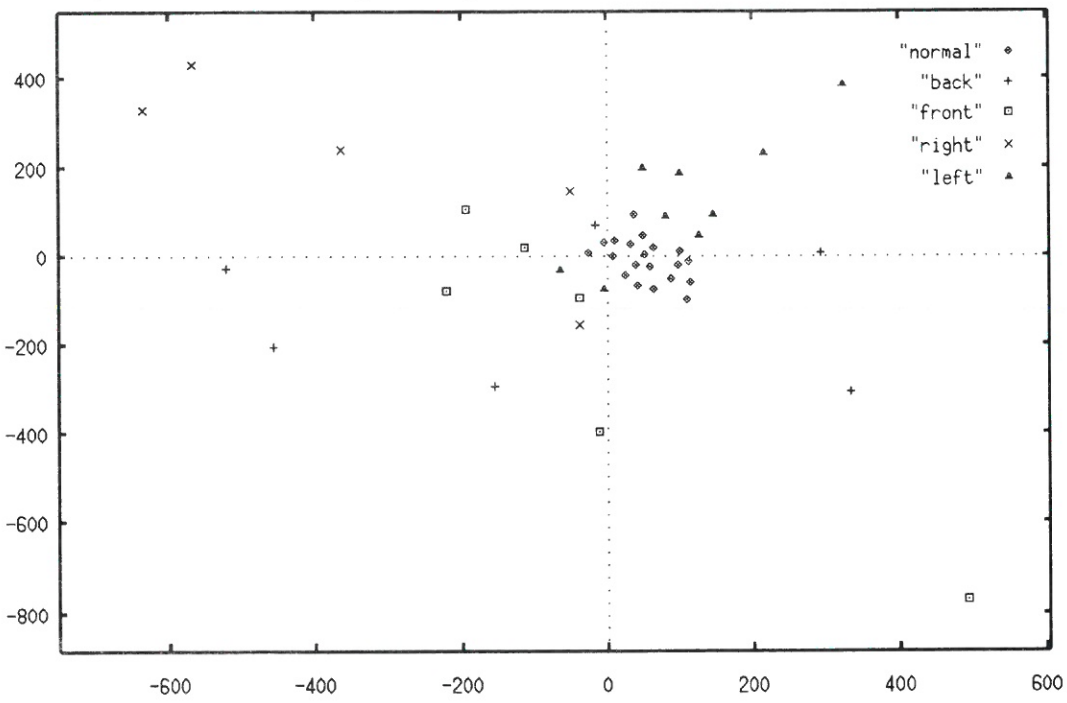


Figure 6. Sammon Plot of the Training Data Available in Problem LP-2 (Transfer)

Table I. Accuracy Results for Several Feature Construction Strategies.

			Learning Problems											
			LP-1			LP-2			LP-3			LP-4		
Number of Classes			4			5			3			5		
Number of Examples			88			47			117			164		
Algorithm			SKIL	Q*	1-NN	SKIL	Q*	1-NN	SKIL	Q*	1-NN	SKIL	Q*	1-NN
Strategy		No. of Features	Accuracy Results (%)											
from 6 traces	F1	90	75.4	88.6	86.4	48.3	68.1	68.1	74.8	88.1	88.0	57.1	68.3	68.3
	F2	42	80.1	85.2	87.5	54.3	70.2	70.2	85.7	92.3	92.3	62.0	70.7	70.1
	F3	36	78.8	87.5	85.2	56.8	74.5	72.3	86.8	90.6	91.5	59.5	72.6	70.1
	F4	48	82.5	88.6	88.6	52.9	57.4	57.4	81.7	86.3	86.3	62.4	62.8	62.8
	F1+2	132	81.5	88.6	86.4	48.3	66.0	68.0	65.2	80.0	88.0	66.0	69.5	66.5
	F1+3	126	78.9	88.6	86.4	48.3	68.1	68.1	80.1	88.0	88.0	65.0	70.1	66.5
	F1+4	138	78.0	86.4	89.8	44.3	70.2	68.1	79.6	88.9	88.9	55.0	68.3	67.1
from 14 traces	F1	210	69.7	90.9	92.1	50.1	66.0	68.1	74.8	86.3	87.2	55.3	69.5	69.5
	F2	84	75.6	90.9	94.3	70.0	74.5	72.3	81.0	91.5	92.3	60.5	75.0	71.3
	F3	112	81.6	89.8	94.3	64.0	76.6	74.5	82.2	88.0	89.7	60.4	78.1	76.2
	F2+3	168	86.8	90.9	92.1	63.4	74.5	74.5	90.9	90.6	89.7	63.3	75.6	74.4
	F1+2+3	378	88.9	90.9	93.2	66.8	63.8	68.1	88.1	88.9	88.9	64.5	72.0	71.3
	F4	112	83.5	86.4	85.2	52.7	63.8	61.7	76.2	83.8	83.8	65.2	64.2	63.4
	F1+4	322	87.5	88.6	89.8	46.4	66.0	66.0	74.4	89.0	89.0	62.2	68.9	68.3
	F1+2+3+4	490	88.9	88.6	90.9	66.8	66.0	68.1	87.3	89.0	89.0	63.0	69.5	70.1

It should be emphasised that a real physical setup (not a simulated scenario) (see Figure 5) was used to collect the training data in these four learning problems.

4.2 Feature Construction

As emphasised above, often there is no model of these sensors or the situations that they help understand. On the other hand, the direct application of learning techniques to the raw sensor data may show limited success in generating such models automatically. The relevant features in each situation may be not directly "visible" in the sensor data. In this Section, several strategies for feature construction are tested on the selected problems. Feature construction, an issue that remains an art, being highly domain-dependent, has not received enough attention from the Machine Learning researchers, traditionally more concerned with feature selection only [4]. For simplicity, each strategy is identified by a code, and the combination of strategies is identified by listing the codes of each of them.

F1: Use directly the values in the force and torque traces.

F2: It would be desirable that the Execution Supervisor could reason about the evolution of force and torque values, measured during the execution of actions, in terms of its overall characteristics, and not in terms of the individual numerical values, i.e. in short, as humans do. A human, making a qualitative description of such behaviour, would probably divide it into intervals, and would mention roughly how long these intervals were, which were the average values of each interval, as well as the average derivatives. Dealing with time intervals is not an easy task, mainly when the goal is to apply existing machine learning algorithms to generate new knowledge. Strategy F2 consists of dividing each trace into three sub-intervals (five measures each), each of them calculating averages and slopes. For each trace, a monotonicity measure, defined as the number of increases in consecutive samples in that trace minus the number of decreases, is also obtained [17].

F3: Similar to strategy F3. Instead of three, each force or torque trace is divided into five sub-intervals.

F4: In the four selected learning problems, the failure situations to be learned are all related

to different types of collisions of the robot arm with its environment. When a collision occurs, the arm enters a state of abnormal vibration that disappears after some units of time. Although the corresponding force and torque signals are not periodic, their processing by the Fast Fourrier Transform (FFT) could provide information in a useful form. It is necessary to assume that the collected traces of 15 samples represent the interval of periodic waves. For each trace, the FFT produces eight harmonics of which only the frequency modules are used as features for learning.

As already mentioned, each example is composed of six force and torque traces: Fx, Fy, Fz, Tx, Ty and Tz. The example description can be enriched by eight additional traces, namely the traces of forces and torques in 2D and 3D: Fxy, Fyz, Fxz, Fxyz, Txy, Tyz, Txz and Txyz.

The four training sets were processed according to the described feature construction strategies. In the first experiments, only the measured traces were used (6 traces per example). Similar experiments were conducted using both the measured traces and the calculated traces in each example (14 traces in all). Estimation of the classification accuracy was then made by the leave-one-out test, using the three considered algorithms, SKIL (in this case without any hierarchical decomposition), Q* and 1-NN. The accuracy results are presented in Table I.

The results are heterogeneous and not fully satisfactory. The worst results were obtained with problem LP-2, probably because of the lack of examples (see, in Figure 6, a 2D Sammon plot of the 47 examples in this problem and the classes they represent). Globally speaking, the best results were obtained in problems LP-1 and LP-3: accuracy around 80% in the case of SKIL and 89% in the case of Q* and 1-NN.

To evaluate the potential of the feature construction strategies, the obtained results are compared to the first row in Table I, that corresponds to directly using the measured force and torque values ("no processing"). As far as SKIL is concerned, there seems to be no big difference between strategies F2, F3 and F4. In average, each of them produces an improvement of around 6% when compared to "no processing". However, simultaneously applying different strategies can produce improvements of more than 12%.

The combined use of strategies F1, F2 and F3 on the measured and calculated traces, corresponding to 378 features, provided the best result in problem LP-1: 88.9%. Applying strategy F2 to all traces (84 features) provided the best result in problem LP-2: 70.0%. The combined use of strategies F2 and F3 on all traces (168 features) provided the best result in problem LP-3: 90.9%. In problem LP-4, the best result was obtained with strategies F1 and F2 on the measured traces only (132 features) provided the best result. The idea of considering the resulting traces of forces and torques in 2D and 3D (bottom half of Table I) did not lead to any improvement or loss in accuracy.

The results obtained with Q* and 1-NN are generally better than those obtained with SKIL. In the case of "no-processing", the difference between SKIL and the other two is 15%, on average. Q* and 1-NN seem to be quite successful in searching useful information in the data. That explains why these two systems do not benefit so much from constructed features. While SKIL can improve more than 12%, on average, after feature construction, the other two improve 5% at most.

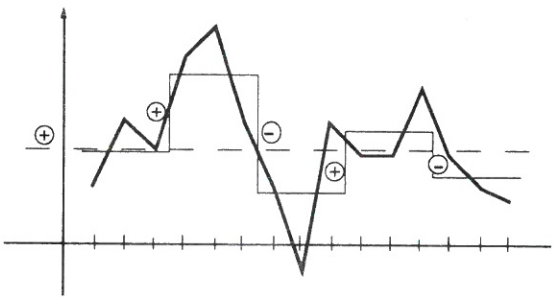


Figure 7. Qualitative Features in A Force or Torque Trace, Used in Assessing Vicinity Between Examples (in this case + + - + -)

4.3 Example Interpolation

One conclusion that can be drawn from other experiments in the assembly domain (see [17]) is that, in general, the classification attributes (= classes) that are better learned — i.e. with greater accuracy — are those represented with more examples in the training set. The negative impact of the lack of examples on accuracy can be reduced by appropriate feature construction strategies and by control of classification errors inside the algorithm itself. However, one question did naturally arise: would it be viable to generate additional examples, by some adequate interpolation method, that enabled more accurate generalization?

An experiment, in what can be named example interpolation, was conducted. For interpolation it is necessary to define: a) a criterion for example vicinity; and b) a procedure for generating a new example from two other neighbour examples.

Table II. Accuracy Results for Several Feature Construction Strategies, Using Interpolated as well as Original Examples

			Learning Problems											
			LP-1			LP-2			LP-3			LP-4		
Number of Classes			4			5			3			5		
Number of Examples			178			86			263			359		
Algorithm			SKIL	Q*	1-NN	SKIL	Q*	1-NN	SKIL	Q*	1-NN	SKIL	Q*	1-NN
Strategy		No. of Features	Accuracy Results (%)											
from 6 traces	F1	90	89.6	88.6	90.9	80.5	83.0	83.0	92.1	95.7	96.6	84.7	88.4	90.2
	F2	42	82.7	92.1	89.8	85.1	78.7	78.7	96.0	97.4	97.4	88.0	85.4	86.0
	F3	36	85.8	88.6	88.6	78.4	80.9	80.9	93.4	96.6	97.4	89.4	86.0	85.4
	F4	48	92.7	90.9	90.9	70.4	63.8	63.8	93.6	93.2	95.0	88.2	75.6	78.1
	F1+2	132	85.2	89.9	90.9	80.5	83.0	83.0	95.7	96.6	96.6	86.1	88.4	89.0
	F1+3	126	84.5	89.9	90.9	80.5	85.1	83.0	96.6	97.4	96.6	85.6	88.4	89.2
	F1+4	138	89.9	87.5	90.9	80.5	78.7	78.7	94.0	95.7	97.4	89.4	87.8	87.8
from 14 traces	F1	210	83.0	94.3	95.5	80.7	80.9	83.0	92.4	95.7	95.7	84.7	88.4	90.9
	F2	84	85.8	94.3	96.6	78.5	83.0	85.1	91.9	96.6	97.4	87.3	89.6	87.8
	F3	112	90.7	94.3	96.6	75.1	85.1	85.1	92.9	96.6	97.4	87.7	89.6	92.1
	F2+3	168	94.2	94.3	95.5	73.6	85.1	85.1	97.4	97.4	97.4	92.3	89.6	90.9
	F1+2+3	378	94.2	94.3	96.6	76.8	80.9	82.3	95.4	95.7	95.7	92.9	90.9	91.5
	F4	112	92.5	88.6	89.8	69.0	68.1	72.3	93.7	94.9	94.9	87.1	81.1	81.1
	F1+4	322	92.2	94.3	95.5	80.7	80.9	78.7	92.2	97.4	96.6	88.3	86.6	87.2
	F1+2+3+4	490	94.2	94.3	95.5	76.8	78.7	78.7	95.7	97.4	96.6	91.6	86.6	87.8

In these experiments, two examples are considered neighbours if they belong to the same class and at least one of the force and torque traces is qualitatively similar in both examples. In this context, two traces are said to be qualitatively similar if the averages of values in each of them have the same sign, and the slopes between the first group of three values and the second one, the second and the third, etc., also have the same signs (see Figure 7). For each example in the original training set, neighbour examples will be searched. An interpolated example is generated for each pair of neighbour examples. Interpolation is done as follows: when traces are similar in both examples, the trace in the new example is the average of the other two; otherwise, the trace in the new example will be directly inherited from the example that motivated the search.

The described example interpolation method was applied to the four data sets used in the previous Section, expanding their size between 83% and 125% (see top of Table II). The new data sets were processed using the feature construction strategies of the previous Section. Finally, SKIL, Q* and 1-NN were applied to the processed data in each problem, producing accuracy statistics (Table II). It should be noted that, in this case, the leave-one-out accuracy test is performed only on the original examples and not on those generated by interpolation. Modifications in SKIL and in the TOOLDIAG package had to be made to account for this requirement.

The results obtained with this approach are much more satisfactory. In what concerns SKIL, accuracy is around 90% in most of the experiments with problems LP-1, LP-3 and LP-4. In problem LP-2, which still has a smaller number of examples, accuracy is around 75% to 80%. The combined use of strategies F2 and F3 on all traces (168 features) provided the best results in problems LP-1 (94.2%) and LP-3 (97.4%). Applying strategy F2 to the raw traces (42 features) provided the best result in problem LP-2: 85.1%. The combined use of strategies F1, F2 and F3 on all traces (378 features) provided the best result in problem LP-4: 92.9%.

These experiments prove that the original training sets contain information that SKIL, despite all the extracted features, could not find. Even Q* and 1-NN, that proved to be more successful in extracting the information relevant to the problem at hand, benefited significantly from example interpolation. The improvements due to interpolation alone (without any feature construction) were, on average, of 22.8% with SKIL, 10.6% with Q* and 12.5% with 1-NN. If the performances of the three algorithms on the original unprocessed training sets were

heterogeneous, being less interesting, the performance of SKIL, now, after interpolation, the results are equivalent: average estimated accuracies on the four learning problems are of 86.7% with SKIL, 88.9% with Q* and 90.2 with 1-NN (see summary in Figure 8). Therefore, it seems that interpolation enabled the algorithms to approach the limit of what is possible to get from the used data.

The SKIL algorithm works much like ID3 [13]. Its advantages are in handling numerical data and generating hierarchies of structured concepts. Like ID3, SKIL cannot take into account error estimation information when choosing a test feature for a given decision node. That is one of the reasons for its great efficiency: for a problem as large as LP-4 in the last row of Table II (359 training examples, 164 test examples, 5 classes and 490 features) SKIL performs the leave-one-out test in less than 30 min. To do the same job, Q*, that is also supposed to be very efficient, takes more than 45 min. Generating additional examples by interpolation, and then applying a simple learning algorithm can be an alternative to heavily processing learning algorithms. In principle, interpolation only generates examples that make sense. To do the same job, the learning algorithm will have to spend a lot of time for exploring uninteresting alternatives.

Attribute	Attribute Value
behaviour	{ normal, failure }
part_status	{ ok, moved, lost }
failure_type	{ collision, obstruction }
collision_type	{ part, tool, front }

a) Attributes

Attribute	Attribute Value	Enabled Attrib
behaviour	failure	{ failure_type }
failure_type	collision	{ collision_type }

b) Enabling triples

Figure 9. Approach-Ungrasp *Problem Specification*

The left-hand side of each rule specifies a conjunction of conditions on feature values that is enough, according to the training data, to recognize the concept described by its classification attributes on the right-hand side.

Hierarchical information, given to SKIL in the form of attribute enabling triples, guides the induction, without requiring a fixed hierarchical structure. The key attributes of a given concept are determined first, and only then the details are paid attention.

This philosophy leads to higher classification accuracy. For the operation `approach-ungrasp`, 117 examples were collected. The SKIL leave-one-out test returns an accuracy of 85%. If the attribute values are combined to produce a one-level set of classes (no hierarchy), the SKIL leave-one-out test returns an accuracy lower than 60%.

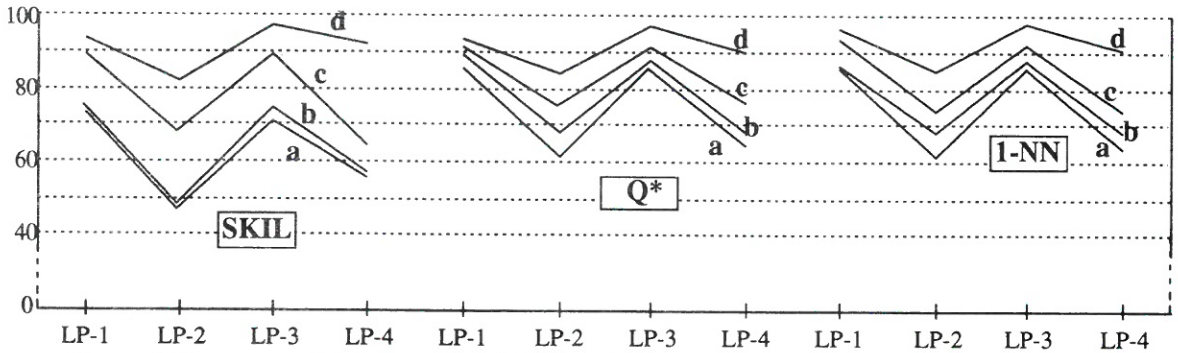


Figure 8. Summary of Accuracy Results Using the Three Algorithms (SKIL, Q* and 1-NN):
a — average of the three worst results; b — no processing; c — average of the three best results without example interpolation; d — average of the three best results with example interpolation

4.4 Multi-level Induction

As we have intuitively shown in [17], multi-level induction is another direction of research that may yield good results in terms of classification accuracy. SKIL was also developed with this problem in mind. Consider, for instance, the domain specification in Figure 9, used to learn the behaviour of the robot during the operation `approach-ungrasp`. The starting attributes are *behaviour* and *part_status*. The behaviour taxonomy generated by SKIL can be translated into a set of rules (Figure 10).

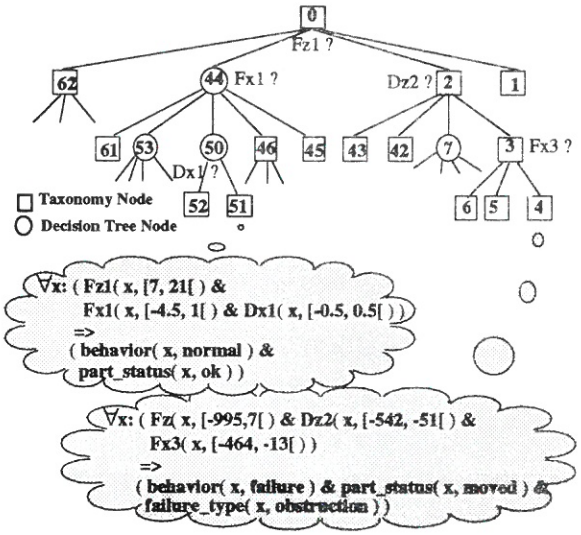


Figure 10. A Fragment of the Taxonomy Generated by SKIL for the *Approach-Ungrasp* Primitive, and Some Corresponding Rules

5. Conclusions

On-line decision- making capabilities must be included in manufacturing systems in order to comply with the new requirements of flexibility and autonomy. Research results concerning execution supervision in flexible assembly systems were presented. The proposed supervision architecture includes functions for dispatching of actions, execution monitoring and failure diagnosis and recovery.

The lack of comprehensive monitoring and diagnosis knowledge in the assembly domain points out to the use of machine learning techniques, leading to an evolutive architecture. The general approach is to collect examples of normal and abnormal behaviour of each operation or operation-type/operator and generate a behaviour model that the diagnosis function will use to verify the existence of failures, to classify and explain them and to update the world model.

Concerning the failure identification /classification problem, the application of the SKIL algorithm, that generates concept hierarchies with a higher degree of accuracy, provided interesting results. Alternative ways to improving accuracy were investigated, namely feature construction and example interpolation. Summary features, like averages and slopes, extracted from sensor traces, made an interesting improvement in accuracy. Frequencies, extracted by the Fast Fourrier Transform, did also contribute to improving accuracy, although to a lesser extent. However, it seems that the measured values (unprocessed) should not be excluded from the learning process. A method for example interpolation, specifically developed for the problems at hand, expanded the training sets, leading to significant improvements in accuracy.

It must be noted that example processing is a highly domain-dependent topic. The example processing methods developed for force-based supervision in assembly can inspire the developing of methods for other domains, but probably will not give the best results if directly applied.

Acknowledgments

This work has been funded in part by the European Union (ESPRIT project B-LEARN II and FlexSys) and JNICT (a Ph.D scholarship and project CIM-CASE). Special thanks go to Manuel Barata, for giving access to his FFT software, and Dirk Tilsner, for an introduction to the TOOLDIAG package.

REFERENCES

1. CAMARINHA-MATOS, L.M., SEABRA LOPES, L. and BARATA, J., **Execution Monitoring in Assembly with Learning Capabilities**, Proc. IEEE Int'l Conf. on Robotics and Automation, San Diego, CA, 1994.

2. DUDA, R.O. and HART, P.E., **Pattern Classification and Scene Analysis**, JOHN WILEY & SONS, New York, 1973.

3. FIKES, R.E., HART, P.E. and NILSSON, N.J., **Learning and Executing Generalized Robot Plans**, ARTIFICIAL INTELLIGENCE, Vol. 3, 1972, pp. 251-288.

4. C. Jutten [coord.] **ELENA Enhanced Learning for Evolutive Neural Architecture**. Deliverable R3-B4-P, ESPRIT 6891, 1995.

5. KAISER, M., GIORDANA, A. and NUTTIN, M., **Integrated Acquisition, Execution, Evaluation and Tuning of Elementary Operations for Intelligent Robots**, Proc. IFAC Symp. on Artificial Intelligence in Real-Time Control, Valencia, Spain, 1994.

6. KANG, S.B. and IKEUCHI, K., **Toward Automatic Robot Instruction from Perception — Temporal Segmentation of Tasks from Human Hand Motion**, IEEE

TRANSACTIONS ON ROBOTICS AND AUTOMATION, 11, 1995, pp. 670-681.

7. KOHONEN, T., **Learning Vector Quantization**, Helsinki University of Technology, Lab. Computer and Information Sci., Technical Report TKK-F-A-601, 1986.

8. KUNIYOSHI, Y., INABA, M. and INOUE, H., **Learning by Watching: Extracting Reusable Task Knowledge from Visual Observation of Human Performance**, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, Vol. 10 (6), 1994, pp. 799-822.

9. MCCARRAGHER, B.J., **Force Sensing from Human Demonstration Using a Hybrid Dynamic Model and Qualitative Reasoning**, Proc. 1994 IEEE Int'l Conf. on Robotics and Automation, San Diego, USA, 1994.

10. MICHALSKY, R.S., **Inferential Theory of Learning: Developing Foundations for Multistrategy Learning**, in R.S. Michalsky and G. Tecuci (Eds.) Machine Learning. A Multistrategy Approach, Vol. IV, MORGAN KAUFMANN PUBLISHERS, San Mateo, CA, 1994.

11. MOZETIC, I., **The Role of Abstractions in Learning Qualitative Models**, Proc. 4th Int'l Workshop on Machine Learning, Irvine, CA, 1987, pp. 242-255.

12. NUTTIN, M., GIORDANA, A., KAISER, M. and SUAREZ, R., **Machine Learning Applications in Compliant Motion**, ESPRIT BRA 7274 B-LEARN II, Deliverable 203, 1994.

13. QUINLAN, J. R., **Induction of Decision Trees**, MACHINE LEARNING , 1, 1986, pp. 81-106.

14. RAUBER, T.W., BARATA, M.M. and STEIGER-GARÇÃO, A., **A Toolbox for Analysis and Visualization of Sensor Data in Supervision**, Proc. Tooldia'93, Int'l Conf. on Fault Diagnosis, Toulouse, France, 1993.

15. RAUBER, T.W., COLTUC, D. and STEIGER-GARÇÃO, A., **Multivariate Discretization of Continuous Attributes for Machine Learning**, in K.S. Barber (Ed.), Proc. 7th Int'l Symp. on Methodologies for Intelligent Systems (Poster Session), Trondheim, Norway, June 15-18, 1993.

16. REICH, Y., **Macro and Micro Perspectives of Multistrategy Learning**, in R.S. Michalsky and G. Tecuci (Eds.) Machine Learning. A Multistrategy Approach, Vol. IV, MORGAN KAUFMANN PUBLISHERS, San Mateo, CA, 1994, pp. 379-401.

17. SEABRA LOPES, L. and CAMARINHA-MATOS, L.M., **Inductive Generation of Diagnostic Knowledge for Autonomous Assembly**, Proc. IEEE Int'l Conf. on Robotics and Automation, Nagoya, Japan, 1995 pp. 2545-52.

18. SEABRA LOPES, L. and CAMARINHA-MATOS, L.M., **Planning, Training and Learning in Supervision of Flexible Assembly Systems**, in L.M. Camarinha-Matos and H. Afsarmanesh (Eds.) Balanced Automation Systems, CHAPMAN & HALL, 1995, pp. 63-74.

19. SUSSMAN, G., **A Computer Model of Skill Acquisition**, ELSEVIER, New York, 1975.