Inductive Generation of Diagnostic Knowledge for Autonomous Assembly

L. Seabra Lopes and L.M. Camarinha-Matos
{lsl, cam}@fct.unl.pt

UNIVERSIDADE NOVA DE LISBOA
Quinta da Torre, 2825 Monte Caparica, PORTUGAL

Abstract

A generic architecture for evolutive supervision of robotized assembly tasks is presented. This architecture provides, at different levels of abstraction, functions for dispatching actions, monitoring their execution, and diagnosing and recovering from failures. Modeling execution failures through taxonomies and causal networks plays a central role in diagnosis and recovery. Through the use of machine learning techniques, the supervision architecture will be given capabilities for improving its performance over time. Particular attention is given to the inductive generation of structured classification knowledge for diagnosis. The applied methodologies, performed experiments and obtained results are described in detail.

1. Introduction

In assembly systems, flexibility has to be achieved by relaxing cell structuration constraints and improving cell control programs in order to adapt to new products/product variations. The operation of industrial robots in flexible production systems, as well as the operation of service robots, pose new problems to the design of their control architecture. In less structured environments, it's difficult to anticipate all possible events. Since it is desirable that the system works autonomously for as long as possible, the control system must be able to make decisions during execution time according to external asynchronous events. In particular, flexible assembly systems will have to cope with execution failures.

The work reported bellow is part of the development of an Execution Supervisor, which receives as input an executable assembly plan and will carry out its execution, performing monitoring, diagnosis and recovery functions. The executable plan is assumed to have a hierarchical structure, which leads to more modular supervision activities. On the other hand, since the planning activity is often carried out hierarchically [6], the proposed plan structure requires no additional effort. From the supervision point of view, the hierarchical approach can be combined with concurrent execution at each level. Plan operators are modeled in STRIPS style.

Modeling execution failures through taxonomies and causal networks plays a central role in diagnosis and recovery. One main problem is the acquisition of knowledge about the environment in order to support monitoring, diagnosis and recovery. For this, the use of machine learning techniques is being investigated. Current results achieved under this approach in the context of the Esprit project B-Learn are described as well as the planned extensions and main foreseen difficulties.

2. Autonomous Assembly Framework

The adopted architecture of the Intelligent Execution Supervisor reflects the hierarchical structure of the plans. For each plan level, its main functions are [4,14,5]:

Dispatching and Global Coordination — A flexible assembly cell can be viewed as a multi-agent system, showing the typical problems of concurrent and asynchronous systems. In the global coordination activities we include: dispatching actions to executing agents, driven by the scheduled plan; synchronization of activities and synchronization with external events; world model update; information exchange; coordination of the other functions of the Intelligent Supervisor and the interaction with the human operator.

Monitoring of Assembly Plans — The monitoring function is used to detect non-nominal feedback in the system during plan execution. Two monitoring modes are usually considered: discrete monitoring and continuous monitoring. Discrete monitoring is used to check preconditions before and goal achievement after the execution of operations. Continuous monitoring is used to check sensory conditions during action execution.

Failure Diagnosis — When the monitoring function detects a deviation of the plan execution from the nominal behavior, the diagnosis function is called. This function performs failure confirmation and update of the internal model. Then the diagnosis function will try to classify and explain the failure. At each execution level, different levels of explanation for a detected failure may be generated, depending on the amount of information available.

In [14] it was proposed to divide errors in 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 cell environment that may cause execution failures. System faults are abnormal occurrences in the assembly resources hardware and software and in communications.

Failure Recovery — At each supervision level, the recovery function is called when the diagnosis function confirmed a failure and found an explanation. The recovery

IEEE International Conference on Robotics and Automation
0-7803-1965-6/95 $4.00 ©1995 IEEE

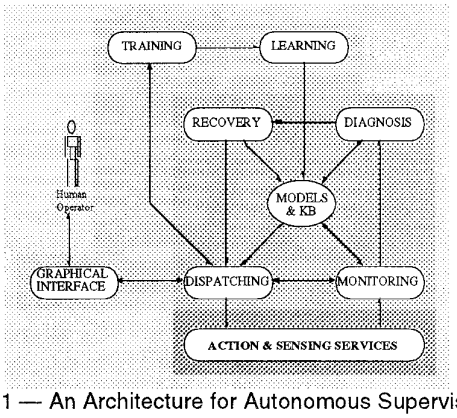


Fig. 1 — An Architecture for Autonomous Supervision

function will try to determine a recovery strategy to bring the execution to a nominal state.

Hierarchical Supervision — In the hierarchical approach proposed by [5], when a failure is detected before, during or after the execution of an action, and it is not possible to classify, explain or recover from that failure, the problem is passed on to the next upper level where context information is broader. In the lower planning levels, recovery actions will tend to be simple reflexes, while in the upper levels determining recovery actions will require more extensive diagnosis and planning.

Training and Learning — The problem of building the knowledge base, and in particular the models that the monitoring, diagnosis and recovery functions need, is not yet solved. 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. 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. Thus, we include in the Execution Supervisor two other functions: Training and Learning (Fig. 1). The training module coordinates the interaction with the human operator in order to acquire new information about nominal execution of the assembly plans as well as descriptions of new error situations. The learning module compiles raw data generating classification knowledge, generalizes instances of target concepts, etc., in order to build the needed models.

3. Generation of Diagnostic Knowledge

3.1. Training or Tutoring Methodology

According to the paradigm of Programming by Human Demonstration, complex systems are programmed by showing particular examples of its desired behavior and giving explanations for particular failure situations. In our current approach, the interaction between the Execution Supervisor and the human operator is fundamental. The human will carry out an initial training phase for the nominal plan execution. The traces of all testable sensors will be collected during training in order to generate the corresponding monitoring knowledge. In the existing implementation, for each action and each continuous feature, the typical behavior of the attribute during the execution of the action is calculated as being all the region between the average minus standard deviation behavior and the average plus standard deviation behavior. The trace of discrete features is also recorded. Also in the initial training phase, the human operator may decide to provoke typical errors, in order to collect raw data in error situations. Error classification knowledge is subsequently generated by induction. When a new failure is detected during real execution of the assembly system, the human operator is called to classify and explain that failure and to provide a recovery strategy for the situation. This is considered also as a training action, since the System History and the model of errors will be expanded and new knowledge will eventually be generated by incremental induction, therefore improving future system performance.

3.2. A Qualitative Reasoning Perspective

In some early approaches to error recovery in robot programs it was already understood that the use of good domain knowledge was important but should be combined with some sort of common-sense and qualitative reasoning [9]. For instance, it is very difficult to model friction mathematically. Still, humans, using their fuzzy understanding of the physical world, can deal with friction in every-day tasks. There is already a very important body of literature coming from the area of Qualitative Physics/Reasoning. However, being quite interesting the available results, they are at the same time a little disappointing, since codifying qualitative knowledge about the physical world turned out harder than expected.

The fact that many of the test cases analyzed by researchers in this area are continuous processes (for instance in chemical plants) seems to have taken them to believe that things change smoothly in the physical world. However, in the robot assembly domain, this assumption does not hold. Errors occur unexpectedly, causing system parameters to change abruptly. Furthermore, the overall nominal execution of an assembly plan cannot be considered a continuous process. At most, some of the primitive actions in the plan can be considered to have a certain degree of continuity. And yet, the execution of an assembly plan is certainly a physical process that, at a certain level of abstraction, should be possible to describe qualitatively. In the following we give an overview of our current ideas about how the model of errors should look like and how to use it, then describe ways to obtain qualitative representations of numerical sensor data relevant to monitoring and diagnosis, and finally present the learning techniques used.

3.3. Model of Errors

Depending on the available sensor information, a more or less detailed classification and explanation for 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 modeling errors in terms of taxonomic and causal links aims at handling this complexity.

In actual execution, when a failure is detected, the current state of the world is analyzed, as well as its evolution in an interval surrounding the time of detection, and a failure classification is determined. This can be done using knowledge generated by inductive machine learning techniques, as will be described bellow. Then, the model of errors will be used to explain the failure, i.e., to determine its causes. Diagnostic reasoning and causality have been studied for some time, and tested frequently in domains like electronic circuits, but there is no unified theory for these matters. Moreover, approaches like the one presented by de Kleer, *et al.* [8] structure the problem considering that the main goal is to determine the faulty components in a system. However, in the assembly domain, not only system faults, but also external exceptions can be causes of off-nominal feedback.

3.4. From Signals to Symbols

The main source of numerical data in our experimental setup is the force & torque sensor. With this sensor we can monitor actions in which the robot arm is involved. 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.

In the field of qualitative physics, the representations for numbers, proposed until now, include signs, inequalities and orders of magnitude. Fuzzy logic could be used to model qualitative values of numerical variables. For instance, consider the behavior of a sensor variable during a certain period of time. A human, making a qualitative description of such behavior, would probably divide it into intervals, and would mention roughly how long these intervals were, which were the average values in each interval, as well as the average derivatives. Thus, fuzzy descriptions for time intervals, amplitudes and derivatives are needed. These descriptions can be given or learned.

Dealing with time intervals is not an easy task, mainly when the goal is to apply existing machine learning algorithms to generate new knowledge. Currently, we divide numerical sensor data behaviors in a fixed number and equal length sub-intervals. For each of them we calculate averages, slopes, etc. The generic approach is to calculate, from the raw sensor data, features closer to the way humans think. The second step will be to calculate a symbolic description of these higher level features.

Since one of the goals is to generate classification knowledge about execution failures, a method was

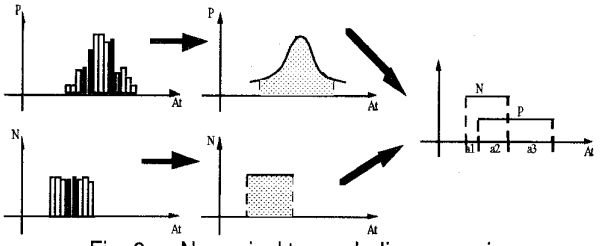


Fig. 2 — Numerical to symbolic conversion.

developed [13] to generate symbolic descriptions of numerical attributes which maximize their class discrimination power. This method follows three steps. The first step is to produce histograms for all pairs of classes and attributes. Each histogram shows the number of objects of each class corresponding to several intervals of values of the attribute. The number of intervals considered is given by the number of Struges: $Interv = 1 + log_2 Tot$, in which Tot is the total number of objects or concepts represented in the diagram, i.e., the total number of objects having that class. In the second step, each histogram will be approximated to several well known statistical distributions (e.g. normal, exponential, uniform). The chi-square test will determine which distribution fits better in the histogram. The last step is to apply a rate of significance to the distribution in order to ignore values of the attribute that do not occur significantly in the objects of the considered class. Finally, the intersection of the intervals of values of the attribute in which each of the classes may occur, defines qualitative values for that attribute (Fig. 2).

3.5. The CONDIS Learning Kit

For the first experiments concerning the assembly supervision domain, an inductive learning algorithm, reported in [10], was applied [5]. The algorithm is simple and, compared to ID3, AQ and other well know symbolic inductive learning algorithms, has the advantage of dealing elegantly with numerical training data. The disadvantage is that it does not consider discrete features. The package CONDIS (inductive learning in continuous and discrete domains), developed in-house, may be used in domains characterized by symbolic and/or numerical features. It was designed to be easily integrated in an application. It can be used to test different approaches to a particular learning problem.

The objects in a domain are characterized by a set of attributes or features, F, and can be grouped according to a set of classes, C. Continuous features take numerical values and discrete features take one of a finite pre-specified set of values. The calculation cost for each attribute may also be provided.

As pointed out by Cheng *et al.* [7], a symbolic inductive learning algorithm is a recursive procedure for which four rules must be specified:

1. TestStop: a rule for deciding when to stop the recursion, i.e., when to make leafs.
2. Classify: a rule for labeling a leaf with a class.

```
algorithm CONDIS(LEx,LCl,LFt) {
// LEx, LCl, LFt — Lists of examples, classes and features
  InitInduction(LEx,LCl,LFt);
  return INDUCTION(LEx,LCl,LFt);
}
algorithm INDUCTION(LEx,LCl,LFt) {
  declare Node;
  if TestStop(LEx, LFt) {
    Node.type = LEAF;
    Node.class = Classify(LEx);
    return Node;
  }
  Node.type = TEST;
  TransformFeatures(LEx,LCl,LFt);
  TF = SelectFeature(LEx,LCl,LFt);
  for each TFk in (TF.transformed_values) do {
    NewLEx = PartitionExamples(LEx,TF,TFk);
    NewLCl = { all classes in NewLEx };
    NewLFt = LFt - TF;
    Node.sub_tree[k] =
        INDUCTION(NewLEx,NewLCl,NewLFt);
  }
  return Node;
}
```

Fig. 3 — The CONDIS Inductive Kernel

3. SelectFeature: a rule for selecting a test feature.
4. PartitionExamples: a rule for partitioning a set of examples.

In addition, for domains characterized by complex features, a rule for transforming the set of values that they can take [TransformFeatures] is considered. Considering a rule for the initialization of the induction process (e.g. for data structure initialization or for feature pre-processing) [InitInduction] might also be useful. The CONDIS learning kit, implemented in C for efficiency reasons, allows for the definition of such rules. For example, if the needed rule implementations are provided, CONDIS can work as a classical ID3. The inductive kernel of the system is described by the pseudo-code in Fig. 3.

3.6. The SKIL Algorithm

The most widely known empirical inductive learning algorithms, such as ID3 [12], AQ [11] and CART [2], are only able to learn "flat" concepts, uni-dimensional concept descriptions, or "labels": the resulting knowledge is only able to assign classes to objects from a given domain. One extension could be an algorithm that learned a multi-dimensional concept description.

However, having as motivation the automatic construction of the models required for the Assembly Supervisor, the idea of generating a concept hierarchy became more attractive. A new algorithm, SKIL (Structured Knowledge generated by Inductive Learning), was developed to perform that task. SKIL requires the following specification of the application domain:

The concepts in the hierarchy are characterized by a set of classification attributes: $A = \{ A_i : i = 1..a \}$. Each attribute can take one of a set of discrete values: $Values(A_i) = \{ A_{ij} : j = 1 .. vA_i \}$. The structure of the most abstract concepts in the hierarchy will be given by attributes selected from a set of top-level (start) attributes, which must be provided by the user: $TLA \subseteq A$.

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 may make sense to calculate other attributes. Therefore, the user should 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 include attributes in EA_{ij} in the set of attributes to be considered in the continuation of the induction process. SKIL assumes that the domain knowledge provided in this way is appropriate.

The values of the attributes of the concepts in the hierarchy are determined inductively based on training data specified in terms of a set of discrimination attributes or features: $F = \{ F_i : i = 1 .. f \}$. When features are continuous, qualitative values are obtained in each step of the induction using the method described in §3.4. For discrete features, the set of values is provided by the user: $Values(F_i) = \{ F_{ij} : j = 1 .. v_{Fi} \}$. 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 as outlined in Fig. 4. Several steps are implemented as separate functions:

- OpenAttributes(LEx,LAt,LAET) — [Closed attributes are those that have the same value in all the examples in LEx. For the open attributes the value cannot yet be determined at the current node]. Closed attributes are removed from LAt. Recursively, add new attributes

```
algorithm SKIL(LEx,LAt,LAET,LFt) {
  // LEx, LAt, LFt are lists of examples, attributes and
  // features. LAET is the list of attribute enabling triples.
  declare Node;
  NewLAt = OpenAttributes(LEx,LAt,LAET);
  Node.closed_ats = ClosedAttributes(LEx,LAt,LAET);
  if TestStop(NewLAt,LFt) {
    Node.type = (NewLAt == LAt ? LEAF : H_LEAF );
    // H_LEAF, a concept hierarchy leaf. LEAF, a tree leaf.
    return Node;
  }
  TransformFeatures(LEx,LAt,LFt);
  (At,TF) = SelectTestFeature(LEx,NewLAt,LFt);
  if (FeatureIrrelevance(LEx,At,TF) > MAX_IRREL) {
    MAX_IRREL — Max. feature irrelevance, e.g. 97.5%
    Node.type = (NewLAt == LAt ? LEAF : H_LEAF );
    return Node;
  }
  NewLFt = LFt - TF;
  for each TFk in (TF.transformed_values) do {
    NewLEx = PartitionExamples(LEx,TF,TFk);
    Node.sub_tree[k] =
    SKIL(NewLEx,NewLAt,LAET,NewLFt);
  }
  Node.type = (NewLAt == LAt ? TEST : H_NODE );
  // H_NODE, a concept hierarchy node. TEST, a decision.
  return Node;
}
```

Fig. 4 — The SKIL Algorithm.

following the attribute enabling triples, LAET. In each step, the added attributes that are closed are removed.

- ClosedAttributes(LEx,LAt,LAET) — Returns the list of closed attributes, among all the attributes that can be reached through LAET, starting in LAt.
- TestStop(LAt,LFt) — Returns TRUE if either LAt or LFt is empty. Otherwise FALSE.
- TransformFeatures(LEx,LAt,LFt) — Obtain qualitative values for all numerical features in LFt, following the procedure outlined in §3.4.
- SelectTestFeature(LEx,LAt,LFt) — For all pairs attribute-feature calculate entropy of classification of the feature concerning the attribute. Return the pair with lowest entropy.
- FeatureIrrelevance(LEx,At,TF) — Returns a confidance factor (chi-square test for stochastic independence [12]) on the irrelevance of test feature TF concerning the discrimination of the values of attribute At.
- PartitionExamples(LEx,TF,TF_k) — Returns the list of examples in LEx with value TF_k for test feature, TF.

4. Experimental Results

4.1. Training Situation and Training Data

The experimental work that will be presented concerns the identification of the execution failure, i.e., the classification part of the diagnosis process. Some failures can easily be identified by simple discrete sensors. For instance, if the wrong tool is attached to the robot, that can be detected by one sensor. If the part is missing in feeder, that may as well be detected with little effort. Such kind of knowledge can be easily coded by hand as rules.

However, how to characterize the 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, what would be desirable is that the system learned to look at the force profiles in order to identify different situations.

The chosen situation is the macro-operation «Pick and Place» of a part, which can be hierarchically decomposed as shown in Fig. 5. For the experiments, we selected three of the primitives involved in the operation: a) approach to grasp position (`Approach-Grasp`); b) `Transfer` (of part); and c) approach to the final position (`Approach-Ungrasp`). During the training phase, each of the selected

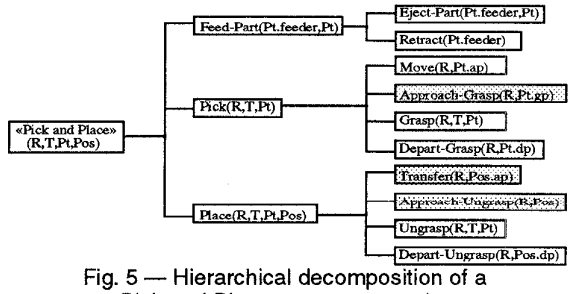


Fig. 5 — Hierarchical decomposition of a «Pick and Place» macro-operation.

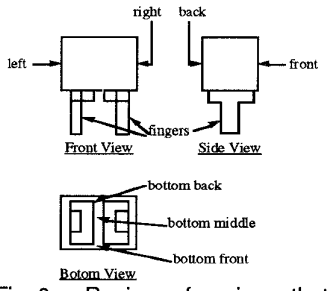


Fig. 6 — Regions of a gripper that may be involved in a failure.

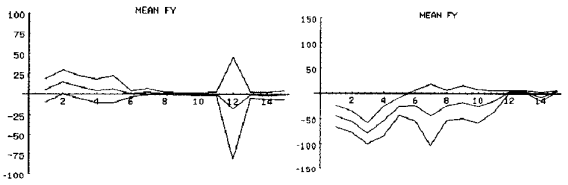
operations was executed many times and several external exceptions were simulated. In most cases an object was placed, either in motion or stopped, in the robot arm motion path. The forces and torques trace in an interval surrounding each failure was collected and the failure classification was associated to it. The length of the trace is of 15 samples.

For the operation `Approach-Ungrasp`, 117 classified examples were collected. The following failure classes were considered (see the instances of force profiles in Fig. 7): 0. normal behavior; 1. collision in part and part moved; 2. collision in part and part lost; 3. collision in tool; 4. front collision and part moved; 5. front collision and part lost; 6. obstruction. For the operations `Approach-Grasp` and `Transfer`, less examples were collected (88 and 47 respectively), but more information about the failure situation was recorded. This information is organized in terms of the following attributes:

behavior — generic information about the operation behavior; can be normal, collision, front collision or obstruction.
body — what was involved in the failure, e.g., the *part* , the *tool* , the *fingers* (*left* , *right* or *both* fingers).
region — region of body that was affected, e.g., *front* , *left* , *right* or *back* side , *bottom* , ... (see Fig. 6).
object size — size of object causing failure: *small* , *large*.
object hardness — can be *soft* or *hard* .
object weight — can be *low* or *high* .

4.2. Experimental Results with CONDIS

To run learning algorithms, some pre-processing of the raw sensor data is needed. In fact, if all numerical values of a force or torque in a trace, in total 15 values, are given to the learning algorithm, probably it will run less efficiently and the knowledge produced will be less readable and less efficient to use. On the other hand, when humans look at force profiles, they can recognize easily trends and high



a) Collision in part & part lost b) Front collision and part moved

Fig. 7 — Examples of typical force behavior during two different types of failures in Approach-Ungrasp

level features that the learning algorithm will ignore if the training set is not given to it in terms of the "good" features. For these experiments, using measures as average, slope and monotonicity, higher level features were extracted from raw sensor data (same method as in [5]). In this way, for each force or torque profile, we reduced the total number of features from 15 to 7, being 4 of them (slope and monotonicity) clearly of a higher level of abstraction.

The first experiments where performed using CONDIS on the data collected during the execution of the `Approach-Ungrasp` primitive. One of the goals was to evaluate the impact of the method presented in §3.4 for conversion from signals to symbols. The following rule implementations were considered:

InitInduction

IN1: Initialize data structures necessary to the other implemented rules.

TestStop

TS1: Create a leaf node when the current list of attributes is empty, or when all examples belong to the same class.

TS2: Create a leaf node in the conditions of TS1 or if the chi-square test for stochastic independence [12] returns a confidence on the irrelevance of the best feature higher than 97.5%.

Classify

CL1: Label the leaf node with the distribution of classes in the current list of examples.

TransformFeature

TF1: "Blind" discretization: for numerical attributes, the domain of values is transformed into a set of intervals of equal length that are used as discrete values.

TF2: The domain of values of each continuous attribute is transformed into a set of qualitative values following the method described in §3.4.

SelectAtrribute

SA1: Entropy based attribute selection criteria [12].

PartitionExamples

PE1: For each of the transformed values of the selected test attribute create the corresponding partition of the current set of examples.

As can be seen from the table in Fig. 8, the largest tree, corresponding to the simultaneous use of rules TS1 and TF1, has 108 nodes: 63 leaf nodes that represent the learned rules and 45 internal nodes that represent the points of decision. As it was expected, rule TS2, which stops branching when the irrelevance of the test feature is too high, reduces the number of decisions to 29. The number of rules is preserved and therefore the global number of nodes is reduced from 108 to 91. On the other hand, rule TF2 reduces the number of decisions from 45 to 35 and the number of rules from 63 to 56. The global number of nodes is reduced from 108 to 92. The smallest tree is produced when rules TS2 and TF2 are used simultaneously. In that case, the number of decisions is reduced to 16 and the global number of nodes is reduced to 72. The rule TF2 produces good results since the tree becomes more concise. When using TS1, the reduction in the number of nodes produced by TF2 is of 16%. When using TS2, the reduction in the number of nodes produced by TF2 is of 22%. However, rule TF2 seems to lead to slightly higher error rates. This is a problem that must be better investigated. In any case, the gain in simplicity of the generated tree seems to compensate for the loss of accuracy. In the four experiments, the error rates (Leave-one-out test) were very high, around 40% to 46%. This is due, mainly, to not having enough training examples.

TestStop	TS1	TS1	TS2	TS2
TransformFeature	TF1	TF2	TF1	TF2
Number of Tree Nodes	108	91	92	72
Number of Decisions	45	35	29	16
Number of Rules	63	56	63	56
Decisions per Rule	0.71	0.63	0.46	0.29
Error Rate	40%	46%	40%	46%

Fig. 8 — Results of applying CONDIS on the data of `Approach-Ungrasp`

4.3. Experimental Results with SKIL

In the `Approach-Ungrasp` problem, the failure classifications have embedded some sort of hierarchy. For instance, there are three major types of collisions: collision in part, collision in tool and front collision. Some of these still have more refined descriptions. This implicit hierarchy could be used to guide the induction process and possibly reduce the error rates. That is what will be attempted next, using the algorithm SKIL. The concept hierarchy that SKIL is going to learn characterizes the execution situation at different levels of detail. The most detailed descriptions will correspond to the seven failure classifications considered above. The set of classification attributes (in the SKIL sense) shown in Fig. 9a seems to be enough to obtain the most detailed descriptions. The attribute enabling statements are shown in Fig. 9b. The top-level (start) attributes are *behavior* and *part_status*.

The discrimination attributes or features are the same as before, and the same pre-processing was applied. The classifications in the table of examples were decomposed according to the classification attributes.

After running SKIL on the new domain specification and new table of examples, a decision tree was obtained having 71 nodes. The concept hierarchy contained in the tree has 59 nodes, being 10 of them internal nodes and 49 terminal nodes (see Fig. 10) The size of this tree is very similar to the size of the tree generated by CONDIS using rules TS2 and TF2 (72 nodes). This was expected since, in SKIL, numerical to symbolic conversion is done by the same method as in TF2. SKIL also uses the chi-square test

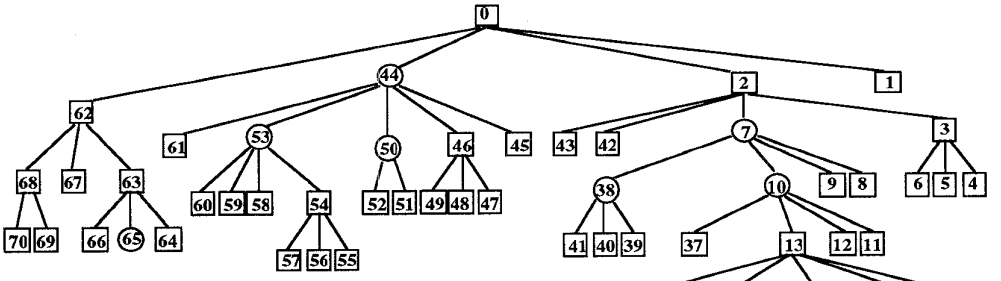
Attribute	**Attribute Values**
behavior	{ normal, failure }
part_status	{ ok, moved, lost }
failure_type	{ collision, obstruction }
collision_type	{ part, tool, front }

a) Attributes

Attribute	**Attribute Value**	**Enabled Attributes**
behavior	failure	{ failure_type }
failure_type	collision	{ collision_type }

b) Enabling triples

Fig. 9 — Approach-Ungrasp problem specification



□ Taxonomy Node
○ Decision Tree Node

Learned descriptions of the target concept at different levels of detail:

Node 0 : { } ;
Node 2 : { behavior = failure } ;
Node 3 : { behavior = failure, part_status = moved } ;
Node 4 : { behavior = failure, part_status = moved, failure_type = obstruction } ;
Node 13 : { behavior = failure, failure_type = collision } ;
Node 24 : { behavior = failure, failure_type = collision, collision_type = front, part_status = moved} ;
Node 51 : { behavior = normal, part_status = ok } ;
....

Fig. 10 — Taxonomy of Execution Failures generated by SKIL for the `Approach-Ungrasp` primitive.

for stochastic independence as in TS2.

Performing the leave-one-out test with the same data and algorithm, the resulting average error rate is 15%, much lower than in the "flat" classification obtained in any of the four experiments with CONDIS (see comparison on table of Fig. 11).

	CONDIS TS2/TF2	SKIL
Number of Tree Nodes	72	71
Number of Taxonomy Nodes	—	59
Number of Taxonomy Leafs	—	49
Number of Taxonomy Interior Nodes	—	10
Number of Decisions	16	21
Number of Rules	56	50
Number of Decisions per Rule	0.29	0.42
Error Rate (Leave-one-out)	46%	15%

Fig. 11 — CONDIS versus SKIL on the data of `Approach-Ungrasp`

From this comparison we see that SKIL may be used to generate more accurate knowledge. However, its great advantage is that it is able to generate conceptual hierarchies. The problem of generating failure classification knowledge for the `Approach-Ungrasp` primitive was initially formulated in terms of seven classes of failures. Then, the problem was reformulated for SKIL in terms of four classification attributes. The total number of complete failure descriptions that can be built using the attributes and their values is 15. Five of these never occur (e.g. { behavior = normal, part_status = lost }) and others correspond to more refined descriptions than those initially considered.

When the user wants to get more and more information about a failure situation, the number of classification attributes and their values increases. If these attribute values are to be combined to produce "flat" classifications or labels, the number of labels increases exponentially, and the problem becomes intractable. This is the case of the information collected during failures of `Approach-Grasp`, which included 10 attributes, 28 values and 8 enabling triples (see domain specification on Fig. 12).

The characteristics of the decision tree and concept hierarchy generated by SKIL starting with top-level attribute *behavior*, are shown in Fig. 13. The global number of nodes is 93. The error rate (30%) is much higher than in the previous problem when SKIL was also applied (15%). This is understandable since the target concept is much more complex and a smaller training set was provided (only 88 examples). In the `Transfer` problem, for which only 47 examples were collected, a taxonomy was also generated by SKIL, and the error rate was 34%. We see, as a general trend, that as the number of occurrences of each attribute value in the training set increases, the corresponding error rate decreases (Fig. 14).

5. Conclusions and Future Work

For a summary of the work concerning the assembly supervision problem, described above, the following aspects may be mentioned:

a) It was finished the implementation of the most important needed features of the experimental setup and integrating infrastructure, namely the implementation of

Attribute	Attribute Values
behavior	{ normal, failure }
phase	{ initial, middle, terminal }
failure_type	{ collision, fr_collision, obstruction }
affected_body	{ tool, tool_tubes, fingers }
tool_region	{ front, right, left, back, bottom }
fingers_region	{ left, right, both }
bottom_subregion	{ front, middle, back }
obj_size	{ small, large }
obj_hardness	{ soft, hard }
obj_weight	{ low, hight }

a) Attributes and values.

Attribute	Value	Enabled Attributes
behavior	failure	{ failure_type, affected_body }
behavior	normal	{ phase }
failure_type	collision	{ obj_size, obj_hardn, obj_weight }
failure_type	fr_collision	{ obj_size, obj_hardn, obj_weight }
failure_type	obstruction	{ obj_size, obj_hardn, obj_weight }
affected_body	tool	{ tool_region }
affected_body	fingers	{ fingers_region }
tool_region	bottom	{ bottom_subregion }

b) Attribute enabling statements

Fig. 12 — Attributes, values and enabling triples.

Number of Tree Nodes	93
Number of Taxonomy Nodes	86
Number of Taxonomy Leafs	62
Number of Taxonomy Interior Nodes	24
Number of Decisions	31
Number of Rules	62
Number of Decisions per Rule	0.50
Error Rate (Leave-one-out)	30%

Fig. 13— Evaluation of knowledge generated by SKIL on the data of `Approach-Grasp`.

robot guarded movements, a feature that is not provided by the controller language. This enabled the implementation of a prototype supervisor capable of action training, execution and monitoring.

b) Definition of an experimental scenario and the realization of experiments and collection of training data.

c) Design and implementation of learning algorithms, CONDIS and SKIL, in order to facilitate integration of learning in the application. With CONDIS we empirically demonstrate the viability of our approach concerning the signals to symbols conversion. The results obtained with SKIL seem rather promising since it produces structured (taxonomic) knowledge with a higher degree of accuracy. In terms of efficiency, both systems run very fast and seem to be capable of handling much larger training sets than the ones used in the described experiments.

d) Accuracy, however, is a problem to be better investigated in the future. Work, in the context of the european Esprit project B-Learn, in cooperation with the University of Turin, in which the learning tool Smart+ [1] was applied to the same data, could not solve the accuracy problem either.

e) If the lack of examples, which are expensive to acquire, was one of the main causes for the less satisfactory results concerning accuracy, this implies that more research effort must be put in the design of the training methodology. Nevertheless, we think that a good approach to the problem should include research in efficient ways to collect examples, in feature construction and selection and in long-term learning. Further results concerning this topic can be found in [16].

Acknowledgments

This work has been funded in part by the European Community (Esprit project B-Learn and FlexSys) and JNICT (projects SARPIC and CIM-CASE, and a PhD scholarship). We thank Mr. João Carlos Silva for his contribution to the experimental setup.

References

[1] Botta, M.; Giordana, A. (1993) — "SMART+: A Multi-Strategy Learning Tool", *Proc. IJCAI-93*, pp. 937-943.

[2] Breiman, L.; Friedman, J. H.; Olshen, R.A.; Stone, C.J. (1984) — Classification and Regression Trees, Belmont, CA.

[3] Camarinha-Matos, L.M. (1989) — *Sistema de programação e controle de estações robóticas - Uma arquitectura baseada em conhecimento, PhD Thesis*, Universidade Nova de Lisboa, 12 Jun. 1989.

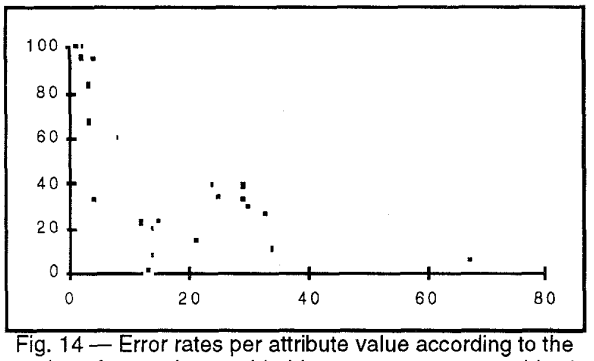


Fig. 14 — Error rates per attribute value according to the number of examples provided (`Approach-Grasp` problem).

[4] Camarinha-Matos, L.M.; Osório, A.L. (1990) — Monitoring and Error Recovery in Assembly Tasks, *23ª ISATA*, Viena.

[5] Camarinha-Matos, L.M., L. Seabra Lopes, J. Barata (1994). Execution Monitoring in Assembly with Learning Capabilities, *Proc. of the 1994 IEEE Int'l Conf. on Robotics and Automation*, San Diego.

[6] Chakrabarty, S.; Wolter, J. (1994) — A Hierarchical Approach to Assembly Planning, *Proc. 1994 IEEE Int'l Conf. on Robotics and Automation*, San Diego.

[7] Cheng, J.; Fayad, U. M.; Irani, K.B; Qian, Z (1988) — Improved Decision Trees: A Generalized Version of ID3, 5th *Int'l Conf. on Machine Learning*, pp. 100-106.

[8] de Kleer, J., A. K. Mackworth, R. Reiter (1990). Characterizing Diagnoses, *The Eighth National Conference on Artificial Intelligence (AAAI-90)*, Boston, 324-330.

[9] Gini, M. (1987). Symbolic and Qualitative Reasoning for Error Recovery in Robot Programs, *NATO ASI Series, Lang. for Sensor-Based Control*, vol. F29.

[10] Hirota, K.; Arai, Y.; Hachisu, S. (1986) — Moving Mark Recognition and Moving Object Manipulation in Fuzzy Controlled Robot, *Control-Theory and Advanced Technology*, vol. 2, no. 3, pp. 399-418.

[11] Michalsky, R.S. (1973) — "Discovering Classification Rules using Variable-valued Logic System VL1", *Proc. of the 3rd Int'l Joint Conf. on Artificial Intelligence*, Stanford.

[12] Quinlan, J. R. (1986) — "Induction of Decision Trees", *Machine Learning* , 1, pp. 81-106.

[13] Seabra Lopes, L. (1991) — *Sistema Integrado de Indução*, Technical Report GR RT-SD-7-91, Univ. Nova de Lisboa.

[14] Seabra Lopes, L., L.M. Camarinha-Matos (1993) — Learning in Assembly Task Execution, *Proc. II European Workshop on Learning Robots* , Turin, Italiy.

[15] Seabra Lopes, L.; Camarinha-Matos, L.M. (1994) — Learning to Diagnose Failures of Assembly Tasks, *Proc. of Int'l Conf. on Artificial Intelligence in Real-Time Control*, Valencia, Spain, October 1994.

[16] Seabra Lopes, L.; Camarinha-Matos, L.M. (1995) — Planning, Training and Learning in Supervision of Flexible Assembly Systems, submitted to *BASYS'95 IEEE/ECLA/IFIP Int'l Conf. on Architectures and Design Methods for Balanced Automation Systems*, Vitoria, Brasil, July 1995.