7

Planning, Training and Learning in Supervision of Flexible Assembly Systems*

L. Seabra Lopes and L.M. Camarinha-Matos
Departamento de Engenharia Electrotécnica, Universidade Nova de Lisboa, Quinta da Torre, 2825 Monte Caparica, Portugal.
Fax: +351-1-2957786. E-mail: {lsl, cam}@uninova.pt.

Abstract

In the context of balanced automation systems, a generic architecture for evolutive supervision of robotized assembly tasks is presented. This architecture provides, at various abstraction levels, functions for dispatching actions, execution monitoring, and diagnosing and recovering from failures. A planning strategy and domain knowledge for nominal plan execution and error recovery is described. Through the use of machine learning techniques, the supervision architecture will be given capabilities to improve its performance over time. The participation of humans in the training and supervision activities is considered essential. The combination of human interactivity with automatic aspects (planning, learning, ..) is discussed.

Keywords

Robotized Assembly, Action Sequence Planning, Monitoring, Failure Diagnosis, Machine Learning, Programming by Demonstration.

1 INTRODUCTION

Manufacturing companies throughout the world are, today, receiving pressures from a variety of sources. The globalization of the economy is leading to the production, within short time frames, of high quality and competitively priced products. Another factor is the increasing complexity of products, resulting from their complex functional definition, from a higher number of product varieties and details, from the need to prevent negative impact in the environment, etc. Even small changes in a product may lead to extensive engineering activities, with a number of improvement cycles. On the other hand, independently of the product end users, manufacturing companies make up a global chain of producers/consumers. A company alone does not have control over the complete cycle of making a finished product. Rather, it must be prepared to handle errors and defects in the partial products, received from suppliers.

* This work has been funded in part by the European Community (Esprit projects B-Learn and FlexSys) and JNICT (project CIM-CASE and a Ph.D. scholarship).

L. Camarinha-Matos et al. (eds.), *Balanced Automation Systems*

A great number of companies are therefore undergoing massive transformations. From the perspective of the product development lifecycle, the approach of Concurrent Engineering aims at reducing its duration by integrating all the related functional inputs and needed expertise as well as facilitating the flow of information among concurrent activities. Other concepts are being applied in industrial practice. For instance, leanness (Lean Manufacturing) is concerned with reducing wasteful activities, unnecessary inventory, long lead times, etc. More ambitious, the concept of Agile Manufacturing is aimed at enabling the production of highly customized products, when and where the client needs them. This certainly involves integrating concurrent activities and implementing leanness, but also redefining the relationship between the enterprise components and the customer.

In terms of the manufacturing resources, many companies, whose production lines are oriented towards a single product, or a small number of varieties, are now struggling with the need to support a larger number of product varieties, often to be produced in small amounts. This is not a trivial problem to solve, since in traditional automation systems, the setup time required to change over to a different product variety is orders of magnitude greater than the cycle time. Usually, production lines are composed of specially designed machines, integrated by materials handling systems. The manufacturing environment is very structured.

The ability to produce highly customized products, in order to satisfy market niches, is a competitive advantage that companies are trying to possess. This requires the introduction of new features in automation systems — flexibility, adaptability, versatility — relaxing cell structuration 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 included: 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 depends, however, on the capacity to handle unforeseen events, which occur in a great number, due to the reduction in system structuration 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, in order to reduce set up costs and assure a rapid response to product innovation, process changes, or shifts in demand; sensorial perception and status identification; on-line decision making capabilities, such as monitoring, diagnosis and error recovery, to cope with the non-deterministic behavior of less structured cells; information integration, viewing the assembly system as part of a more general CIM system.

Of course, it will not be possible to achieve full system autonomy, since it is very difficult to anticipate all possible events and failures at programming time. To be successful, even the most advanced technologies depend on adequate consideration of the human factor.

The work being conducted in our labs, towards an architecture for supervision of assembly and manipulation tasks, considers, at different planning levels, functions for dispatching actions, monitoring their execution, and diagnosing and recovering from failures. The planning approach is initially presented, identifying the interactive aspects and describing the automatic ones. The experimental setup is described with emphasis on the integration effort under a perspective of migration from legacy systems

Our main concern is the acquisition of knowledge about the tasks and the environment to support monitoring, diagnosis and error recovery. A training methodology was devised, in which a human tutor can provide the system with apriori knowledge, with examples of desired behavior and examples of concepts to be learned (programming by human demonstration). Training is considered both at initial system programming and when unanticipated situations arise. Based on the demonstrated examples and using machine learning techniques, the system will incrementally build the needed models.

2 PLANNING AND SUPERVISION

2.1 Planning Framework

Planning in manufacturing and assembly is typically done in a hierarchical fashion, starting in long-term planning activities, at the upper level, and detailing until the necessary executable plans are generated. At the lower planning levels, product and process oriented planning, scheduling at the shop floor and detailed execution planning are considered. Product oriented planning determines a feasible assembly precedence graph taking into account the constraints derived from the product model (bill of materials, geometric model, tolerances model, materials model, etc.). Process oriented planning, taking into account feasibility conditions from the technologic point of view, generates additional constraints, more concerned with the resources (robots, end-effectors, feeders, fixtures, etc.) that must be used.

Several attempts to adapt generic planners (developed in the AI community and having in mind toy problems as "the blocks world"), to realistic robotics tasks have been made. Other approaches are strongly geometric-reasoning-based but require heavy processing procedures and achieved results still present some limitations. On the other side, the most adequate spatial-related solutions are not completely justified by pure geometric reasoning but depend on other technological constraints.

We feel that the two phases of product oriented planning and process oriented planning can more easily be performed in interaction with the humans. Various interactive planning systems (Computer Aided Process Planning — CAPP) have been developed in recent years (Furth, 1988) to help the human expert in generating appropriate process plans. Most of these systems follow an interactive generative approach, in the line of a decision support system, helping the production engineer in the generation of a feasible sequence of assembly steps. The process plan typically includes information on abstract operations to be performed and respective precedences, goal positions, mating referentials, approaching directions, possible part grasping zones, part stable poses, types of resources that can be used in each operation, etc.

Finally, when all the information about an assembly task is gathered and a specific cell is selected, the high-level specification contained in the process plan is instanciated. It is now the moment to plan the execution, i.e., to determine all needed actions, their characteristics and parameters and their optimal sequence.

2.2 Planning Strategy and Representation

For nominal planning, and also for failure recovery, a planner , in the AI sense of the term, was developed. This planner, implemented in Prolog, uses a domain independent planning strategy, but takes into account domain knowledge provided in a pre-defined format. The planning strategy is, basically, a best-first forward search procedure. From the initial state of the world, new states are generated, by applying operators of the considered domain, until the goal state is reached. In each step, a set of legal operator instantiations is determined, and evaluated according to domain dependent heuristics. The operator with highest score is selected for continuing the search. The way the planner uses the provided domain knowledge to select operators makes it a non-linear planner, since it can handle interactions between goals.

The domain specification includes: definition of operators; templates of facts about the world; typical precedences between facts in the goal state; and measures of contribution of facts asserted in a given phase to goal facts. An operator is defined as a Prolog clause having the following format (see example in Fig. 1):

```
operator(Op,Info,Keep,Del,Add).
```

Where:
- `Op` — template of the operator, specifying name and parameters.
- `Info` — list of facts about the world.
- `Keep` — pre-conditions not removed by execution of the operator.
- `Del` — pre-conditions removed by execution of the operator.
- `Add` — conditions that become true as consequence of executing the operator.

In order to handle non-linear planning problems, this planner can take into account known precedence relations between facts in the goal state. The goals not preceded by any other goals in the goal state are selected to be solved first. When each of them is solved, some new goals will be considered by the planner, and so on. In the blocks world, the following precedence relation would be enough: `typical_precedence(on(B,C),on(A,B)).`

In each phase of the planning process, the pre-conditions of all operators will be matched to the current world state. The result is a set of legal operator instanciations whose usefulness, concerning the solution of the goals currently being considered, is evaluated. This evaluation is made taking into account the contributions of each of the added facts to each of the goals currently being considered. This evaluation is, again, domain dependent, and is made based on Prolog rules of the following form:

```
contrib_to_goal(CS,F,G,Score):- <Rule Body>.
```

Basically, the rule says that in a given state `CS`, an operation that asserts the fact `F` will contribute to goal `G` in a way measured by the returned `Score`.

2.3. The Assembly Domain Knowledge

In Assembly, domain knowledge is much more complex than in the blocks world. At the center of the problem are the graphs of precedences for assembly and for disassembly, the graph of connections between components, and of course geometrical data. A good human interface is necessary to acquire all this information. Graphical simulation can be used to acquire information on positioning, grasping, trajectory skeletons, etc. For instance, the needed referentials (grasp and approach positions, mating referentials, etc.) can easily be "suggested" by the human expert by pointing on a screen. The evaluation of the precise values of such referentials can then be done by an automatic function. This leads to a "light" procedure, implementable with available technology (see, for instance, systems like ROBCAD or IGRIP).

The planning strategy, described above, is used to determine sequences of actions, at a sufficiently high level of abstraction. Still, the planner must take into account the positions of parts in feeders and pallets, the mating positions, the mating precedences, the tools for grasping and mating, the needed elementary skills, etc. Examples of an operator, a precedence rule and a contribution rule are presented in Fig. 1. Besides the presented `assemble` operator, the planner can make use of a `disassemble` operator, especially useful in correcting assembly errors, and various operators for picking and placing parts in/from fixtures, feeders or the free workspace, for fetching and storing tools, for feeding parts and pallets, etc.

It is not possible to completely describe here a realistic planning example. Just for illustration, we mention an experiment with the Cranfield Benchmark, a well known laboratory product used for testing in the assembly domain. It is a pendulum composed of seventeen parts: two side plates, four

```
operator(
  assemble_component
       (R,T,Obj,Comp,Part,Prod,Fix,Cp,Geom),
  [% Info:
     object_type(Obj,Part),
     part_tool(Part,T),
     component_contacts(Comp,LComp),
     mate(Prod,Comp,Part,Geom,Prec,Succ)],
  [% Keep-PC:
     current_tool(R,T),
     not(assembled(Comp,Prod,Fix)),
     fixture_with_product(Fix,Prod),
     not(robot_arm_breakdown(R)),
     not(tool_breakdown(T)),
     not(defective(Obj)),
     all(C,Prec,assembled(C,Prod,Fix)),
     all(C,LComp,
        [ assembled(C,Prod,Fix)),
          represented_by(C,X)]
          -> [not(defective(X))]) ],
  [% Del-PC:
     current_arm_position(R,Cp),
     object_in_robot(Obj,R) ],
  [% Add-C:
     assembled(Comp,Prod,Fix),
     represented_by(Comp,Obj),
     robot_free(R),
     current_arm_position(R,Dp) ]
).

typical_precedence(
        assembled(C1,P,F),
        assembled(C2,P,F))
:- clause(mate(P,C2,_,_,Prec,_),true),
   member(C1,Prec).

contrib_to_goal(CS,
  pallet_available(Pal,_),
  assembled(C,Pd,_),
  3) :- member(part_in_pallet(Ob,Pal),CS),
        member(object_type(Ob,Part),CS),
        member(mate(Pd,C,Part,_,LP,_),CS),
        check_prec(CS,C,Pd,LP).
```

Figure 1 Assembly Planning Knowledge.

spacer pegs, a shaft, a lever, a cross bar and eight locking pins. Except for the locking pins, all the other needed mate operations are stack operations, the most common in industrial practice. All mate operations require some degree of compliance. Compliance, however is not handled at the level of action sequence planning. In our experimental setup we have special purpose feeders (which limits cell's flexibility) for side plates and cross bars. The locking pins and the spacer pegs are fed to the system in a pallet and the lever and the shaft in another pallet. Three different tools must be used. Running the planner with the incorporated domain knowledge on this problem produces an optimal plan with 53 operations without any backtracking, which may be considered a good result.

Each of the operations in the generated plan must be expanded, in a more or less deterministic way, into a sequence of resource-level operators, like `move`, `approach`, `transfer`, `peg_in_hole` or `grasp`. This involves lower levels of planning, more concerned, for instance, with trajectories or compliance. A skill acquisition approach to compliance, using learning techniques, seems promising (Kaiser, *et al.*, 1994). However, as we are more concerned with execution supervision, we prefer to simplify the planning functionalities of the lower levels in order to be able to realize a working prototype.

2.4 Supervision Functions

Two plan levels were already mentioned. A hierarchical specification of the mate precedence graph or the learning of macro-operations may originate additional plan levels. A hierarchical plan is an advantage for supervision, since it provides different contexts for error detection and recovery. At the lower levels, error recovery will tend to consist of simple reflexive actions. At the upper levels error recovery will require more extensive diagnostic reasoning and planning. The architecture of an Intelligent Execution Supervisor should reflect the hierarchical structure of the plans. For each plan level, the main functions are (Camarinha-Matos, *et al.*, 1994):

Dispatching and Global Coordination — The global coordination activities performed by a high level controller include: dispatching actions to the executing agents; synchronization of activities among agents and with external events; world model update; information exchange.

Monitoring of Assembly Plans — The monitoring function is used to detect non-nominal feedback in the system during plan execution.

Failure Diagnosis — This diagnosis function will firstly check if there really is a failure (failure confirmation) and update the internal model. Then this 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. For example, a gross diagnostic can be "pick fail". A more detailed diagnostic could be "pick fail due to object sliding". The least detailed explanation would be "deviation detected".

Failure Recovery — At each supervision level, the recovery function is called when the diagnosis function confirmed a failure and found an explanation. The recovery function will try to determine a recovery strategy to bring the execution to a nominal state. One basic question is how to build recovery strategies? Since the detected error is some unexpected (abnormal) event and, therefore, the nominal plan is not to be altered.

3 THE TRAINING METHODOLOGY

One important aspect 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 behavior and the nominal operation behavior. For example, let's 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 Fig. 2, 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, determine a failure classification and explanation. For this function, additional features must be extracted. Diagnosis is a decision procedure that needs a 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 easily 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.

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. 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.

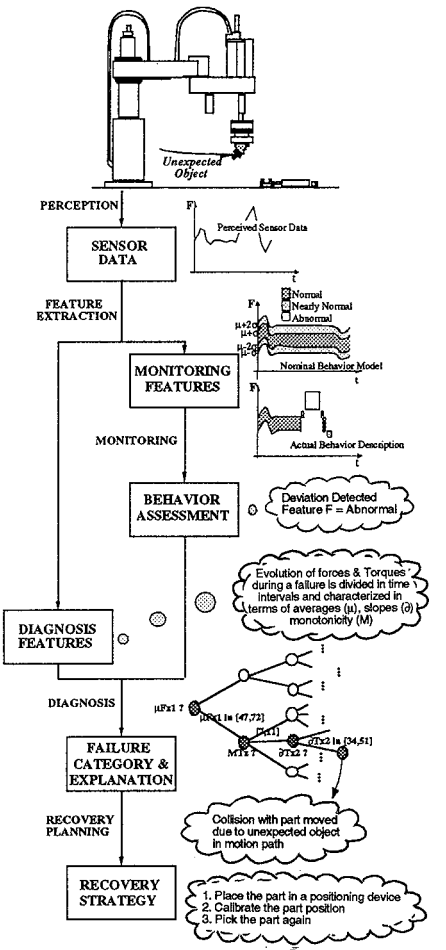


Figure 2 Example of the Error Detection and Recovery Cycle.

For complex operations, that don't 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. In the same way, evaluating the behavior 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. The paradigm of Programming by Human Demonstration (in this case Robot Programming by Demonstration (McCarregher, 1994; Kaiser *et al.*, 1994)) seems indicated to overcome this type of difficulties. According to this paradigm, complex systems are programmed by showing particular examples of their desired behavior. In our approach, interaction with the human, seen as a tutor, is fundamental. Functions for training and learning are included in the supervisor architecture (Fig. 3).

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 actuated 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 use to learn the control function of the operation (application of learning in compliant motion are being investigated by our partners in the B-LEARN project (Nuttin, *et al.*, 1994)).

In which 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 behavior and the average plus standard deviation behavior.

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 were manually provoked and the effects as well as the corresponding traces of sensor values recorded. Examples of provoked external exceptions are: unexpected objects in 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 was generated by inductive algorithms. When flat classifications were generated, the leave-one-out error rate was too high. Taxonomies of failure classifications, generated by SKIL, presented a much better degree of accuracy, but still don't ensure full reliability of the system. Further evaluation of learning techniques is still necessary.

Finally, recovery strategies were programmed for the most common errors. A good user interface should facilitate also the specification of recovery strategies. Feasible possibilities, that we didn't consider, are graphical simulation and virtual reality. In the current prototype recovery strategies are entered in textual form.

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 important to distinguish between training of elementary operations and specific task training (Fig. 3). 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 for the situation, recovery planning is attempted. If a plan is found and, when applied, brings the

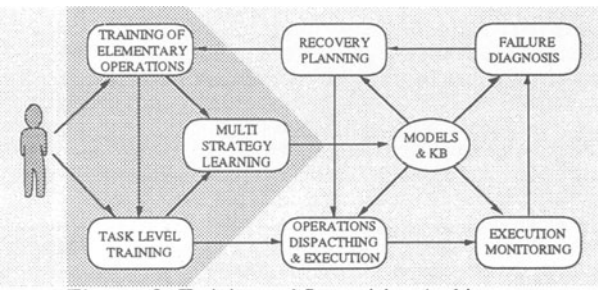


Figure 3 Training and Supervision Architecture

system back to nominal state, this plan will be generalized and stored for future 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 should contribute to the tuning of the related skills (Kaiser, *et al.*, 1994; Nuttin, *et al.*, 1994).

This approach allows for progressive independence of the human operator, who will not need to be permanently supervising the assembly process. On the other hand, as the system relies more on continuous self-evaluation of its behavior 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 form a long-term learning cycle during which the models of the operations, the 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 big impact in the reasoning processes.

4 EXPERIMENTAL DEVELOPMENTS

4.1 Migration from Legacy Systems

Migration from legacy systems is one of the most challenging aspects of manufacturing systems. Existing devices and systems are often too valuable to be thrown away. However, their integration in more advanced supervision architectures poses many problems, some of which we have faced into installing our experimental setup.

The setup being used in the experiments is a robotic assembly cell, composed of one SCARA robot, three robot grippers, magazine and corresponding tool exchange mechanism, two special purpose feeders, one fixture and sensing devices. Needless to say, the robot control language is not suited to write intelligent control software. Moreover, it does not give information about errors and does not provide guarded movements.

As feedback information sources for our supervision system, the following discrete information sensors were integrated in the cell: a) in each gripper, to detect if it is open, closed or clamping; b) in feeders, to detect part presence, part stock existence and feeder problems; c) in the wrist of the robot, to find out which tool is attached, if any; d) in each tool place to detect tool presence; e) in the fixture, to detect if the jig which will hold the assembly is present. The most frequent execution failures are anticipated to be those in which the robot arm is involved, including collisions, obstructions and handling failures. Therefore, a force and torque sensor, which seems a good candidate to give information about those failures, was also included.

A major problem found is that the robot controller does not provide guarded movements. Since communication via Teach Pendant (TP) is quite fast, the solution was to decompose each motion command into a series of incremental steps, executed sequentially via TP, until some condition is verified. A TP Emulator process was used to achieve this goal.

As it is not easy to make acquisition of large quantities of sensorial data in Unix workstations, and, on the other side, concurrency in Unix affects the sensor sampling rate, a program called Low-level Monitor is run in a dedicated PC, where it is quite simple and cheap to implant a data I/O board. This program will check conditions during the execution of actions, as specified by the Intelligent Supervisor, and is able to answer questions about the state of the system during the diagnosis phase. Communication between the Low-level Monitor and the Intelligent Supervisor, running in Unix, is accomplished via an RS232C line.

To integrate and make available the services of sensors and actuators, a Cell Front-End was developed in the Unix environment. This front-end is composed of a set of server processes which interface clients (namely the intelligent supervisor) to the cell resources. An Operational Server process makes the interface with actuators. A Monitor process makes the interface with

sensors via de Low-Level Monitor PC. In general, migration from legacy systems involves integrating existing device controllers in a more flexible programming environment, usually UNIX. Under a client-server approach, the services of the physical devices are mirrored by (possibly a hierarchy of) Unix processes.

4.2 Learning of Hierarchical Diagnostic Knowledge

As emphasized in section §3, 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. Several inductive learning algorithms and systems (e.g. , ID3 (Quinlan, 1986), (Hirota, *et al.*, 1986), Smart+ (Botta, Giordana, 1994), CONDIS (Seabra Lopes, Camarinha-Matos, 1995)) were applied to the assembly domain (Camarinha-Matos, *et al.*, 1994; Seabra Lopes, Camarinha-Matos, 1995). These techniques are only able to learn uni-dimensional concept descriptions: the resulting knowledge is only able to assign classes to objects from a given domain. In the assembly domain, for example, these algorithms and systems cannot handle the problem of discriminating collisions from obstructions and normal situations, handling simultaneously the problem of discriminating between different types of collisions.

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 (Seabra Lopes, Camarinha-Matos, 1995). 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 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 attributes in EA_{ij} should be included in the set of attributes to be considered in the continuation of the induction process. For example, when learning the behavior of a `Transfer` operation, if a collision is found, it may 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 training data is characterized by 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.

To illustrate an application of this algorithm, consider the macro-operation «Pick and Place» of a part and three of the basic primitives involved: 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 operations was executed many times and several external exceptions were simulated. In this case, the goal of applying SKIL is to learn a taxonomy of failure descriptions. The classification attributes and enabling triples used in the `Approach-Ungrasp` primitive were as specified in Fig. 5. The training set was composed of 117 examples. After running SKIL on the domain specification and training set, 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. Examples of the corresponding rules are:

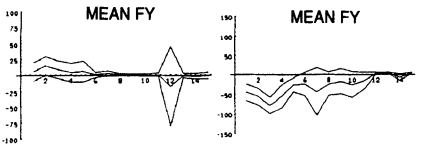
```
∀x: ( Fz1(x,[7,21[) & Fx1(x,[-4.5,1[) & Dx1(x,[-0.5,0.5[) )
     => ( behavior(x,normal) & part_status(x,ok) )

∀x: ( Fz1(x,[-995,7[) & Dz2(x,[-542,-51[) & Fx3(x,[-464,-13[) )
     => ( behavior(x,failure) & part_status(x,moved) &
          failure_type(x,obstruction) )
```

Performing the leave-one-out test with the same data and algorithm, the resulting average error rate is 15%. It should be noted that, using a traditional inductive learning algorithm (such as ID3 (Quinlan, 1986)) and "flat" concept descriptions, equivalent to all combinations of values of the classification attributes, a much higher error rate is obtained.

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` and `Transfer` (respectively 88 and 47 examples):

behavior — generic information about the operation behavior; can be normal, collision, front collision or obstruction; what will be learned is, in fact, a model of the behavior (either normal or abnormal) of the system when performing these operations.
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*, ...
object size — size of object causing failure: *small*, *large*.
object hardness — can be *soft* or *hard*.
object weight — can be *low* or *high*.



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

Figure 4 Typical force behavior in two types of failures in Approach-Ungrasp.

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 Attrib
behavior	failure	{ failure_type }
failure_type	collision	{ collision_type }

b) Enabling triples

Figure 5 The Approach-Ungrasp problem specification.

For the `Approach-Grasp` primitive, the domain specification, provided to SKIL, included 10 classification attributes, 28 attribute values and 8 enabling triples. A decision tree and concept hierarchy with 93 nodes was generated. 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). For the `Transfer` problem, in which only 47 examples were collected, the taxonomy generated by SKIL has an error rate of 34%. We confirm what was expected in this type of domain: as a general trend, when the number of occurrences of each attribute value in the training set increases, the corresponding error rate decreases.

Training the system to understand the meaning of sensor values and learning a qualitative and hierarchical model of the behavior of each operation is a crucial step in diagnosis. Programming such model would be a nearly impossible task.

Since the human defines the "words" (attribute names and values) used in the model, the human is capable of understanding the 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. The failure explanation rules that we are using have the following format:

```
explanation(Op,FD,C,Del,Add) :- ... .
```

The first argument, `Op`, is the elementary operation during which the failure occurred or, in, general the execution context. The second argument, `FD`, is a description of the execution failure, obtained from the sensor data using the taxonomic model. Then, `C` is the ultimate cause of the failure. Knowing the exact cause can be irrelevant. What is important is to identify the state of the system after the failure. The arguments `Del` and `Add` specify what facts must be deleted from the state description and what facts must be added.

For example:

```
explanation(
   get_tool(R,T1,TP1),
   [ [failure_type,wrong_tool],[current_tool,T2] ],
   unknown,
   [ current_tool(R,T1),tool_in_magazine(T2,TP2)],
   [current_tool(R,T2),Fact]
) :- ( toolplace(TPx), Fact = tool_in_magazine(T1,TPx);
       Fact = tool_lost(T1) ).
```

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 in (de Kleer, 1990) 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. The described representation, that we are using, seems to provide useful results, although additional evaluation and refinement are needed.

4.3. Recovery Planning

Finally, when an explanation for an error is obtained, recovery planning is attempted. However, the most common case, probably, is that, not only one, but several possible explanations are found. The available sensor information is not enough to assert, with significant certainty, which is the state of the system. For instance, in the "wrong tool" example, presented above, the explanation rule provides several explanations, namely that the needed tool is in some other toolplace, TPx (and this represents already several possibilities), or lost. In this case, the recovery planner has to assume one of the explanations, generate a recovery plan and use it. If some problem arises, probably the assumption was not correct and some other explanation must be considered.

To be noted is that the initial plan is generated for the nominal conditions. The representation of the operators contains several assumptions. For instance, `not(defective(Obj))` is one of the pre-conditions of the `assemble` operator. Imagine that, in executing the assembly plan of the Cranfield Benchmark, mating a spacer peg with the side plate fails. Various explanations are possible: the spacer peg might be defective, the side plate might be defective, some unexpected object originating in the environment was obstructing the mate operation, etc. Unless some sophisticated sensorial feedback is available, the robot will have to recover based on assumptions or beliefs. The recovery actions will be, simultaneously, verification actions.

One aspect that will be address in the near future is the application of learning techniques in error recovery. A learning feedback loop will have to be implemented (Evans, Lee, 1994). In a first step, when an error, for which no recovery strategy is known, is detected, recovery planning is attempted. If recovery planning fails to generate a plan, eventually the human operator will provide one. In a second step, if the obtained recovery strategy is successful, it will be generalized and archived for future use. This will be attempted in the next phase.

5. CONCLUSIONS AND FUTURE WORK

In the context of balanced automation systems, an evolutive execution supervision architecture was presented. Flexibility implies increasing the on-line decision making capabilities, for which dispatching, monitoring, diagnosis and error recovery functionalites have been devised. 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 participation of humans in the training and supervision activities is considered fundamental in order to achieve flexibility in assembly systems. The general approach is, therefore, to collect examples of normal and abnormal behavior of each operation or operation-type/operator and generate a behavior 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 algorithm SKIL, that generates concept hierarchies with a higher degree of accuracy, provided interesting

results. Further research will focus on efficient generation of examples, feature construction and selection and long-term learning. Developments in planning for nominal execution and for error recovery were presented. The used planning strategy is domain independent, non-linear, forward chaining and depth/best-first. The representation of the assembly domain knowledge was presented. Future research in this topic includes the learning aspects in recovery planning.

6 REFERENCES

Botta, M. and Giordana, A. (1993) SMART+: A Multi-Strategy Learning Tool, *Proc. Int'l Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 937-943.

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

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

Evans, E.Z. and Lee, C. S. G. (1994) Automatic Generation of Error Recovery Knowledge Through Learned Reactivity, *Proc. of the 1994 IEEE Int'l Conf. on Robotics and Automation*, San Diego.

Furth, B. (1988) *Automated Process Planning*, NATO Advanced Study Institute on CIM: Current Status and Challanges, Springer-Verlag, NATO ASI Series.

Hirota, K., Arai, Y. and 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.

Kaiser, M; Giordana, A. and Nuttin, M (1994) Integrated Acquisition, execution, evaluation and Tunning of Elementary Operations for Intelligent Robots, *Proc. of the IFAC Symp. on Artificial Intelligence in Real-Time Control*, Valencia, Spain.

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

Seabra Lopes, L. and Camarinha-Matos, L.M. (1994) Learning to Diagnose Failures of Assembly Tasks, *Proc. of the IFAC Symp. on Artificial Intelligence in Real-Time Control*, Valencia, Spain, October 1994.

Seabra Lopes, L. and Camarinha-Matos, L.M. (1995) Inductive Generation of Diagnostics Knowledge for Autonomous Assembly, to appear in *Proc. 1995 IEEE Int'l Conf. on Robotics & Automation*, Nagoya, Japan.

McCarragher, B.J. (1994) Force Sensing from Human Demonstration using a Hybrid Dynamic Model and Qualitative Reasoning, *Proc. of the 1994 IEEE Int'l Conf. on Robotics and Automation*, San Diego.

Nuttin, M., Giordana, A., Kaiser, M. and Suárez, R. (1994) — *Machine Learning Applications in Compliant Motion*, Esprit BRA 7274 B-LEARN II, deliverable 203.

7 BIOGRAPHIES

Luís Seabra Lopes is a doctoral candidate in the Electrical Engineering department of the New University of Lisbon (UNL), with a grant from JNICT, the portuguese research board. He graduated from the Computer Science department of UNL in 1990, and then became member of the Intelligent Robotics Group of the same university. He is participating in the european ESPRIT project B-LEARN, one of the first projects to investigate applications of Machine Learning to Robotics. At the moment his main interests are Robotics, Machine Learning and Intelligent Supervision.

Luís M. Camarinha-Matos is professor at the New University of Lisbon. He graduated in Computer Engineering from the Computer Science department of UNL in 1979. He then became assistant professor at the same department and participated in several european research projects. He received the Ph.D. degree in Computer Science, topic Robotics and CIM, from the same university in 1989. He was co-founder of the Electrical Engineering department of UNL were he is currently auxiliar professor (eq. associate professor). He also leads the group of Robotic Systems and CIM of Uninova. His main interests are: CIM systems integration, Intelligent Manufacturing Systems, Machine Learning in Robotics.