Error Recovery in Automation - An Overview

Peter Loborg
Department of Computer and information Science
Linköping University, S-581 83 Linköping, SWEDEN
Phone: (+46) 13 282494 E-mail: plo@ida.liu.se

Abstract

This paper attempts to provide an overview of techniques and approaches to error recovery, both in automation and in other fields (such as autonomous robotics) where analogous problems occur. The term 'error recovery' is often used as a common name for the three sub areas error/fault detection, error/fault diagnose, and recovery from the resulting failure. All three areas will be covered.

Rather than presenting different systems and approaches in-depth, different *types* of systems and approaches will be presented and compared.

Terminology

The first observation made is that there is a lack of common terminology, e.g. the word *error* is sometimes used to denote the original reason why something went wrong, whereas in other cases it is used to denote the *effects* in the computer system due to some unforeseen event.

However, in the 'reliable computing' community there is an emerging standard [Laprie, 1992]. Although they are primarily interested in problems internal to a computer and its software, the terminology is applicable when talking about problems in a 'real world' controlled by a computer, problems which may manifest themselves as discrepancies between the actual state in the real world and what the computer 'think' is the actual state. The following definitions will be used throughout this paper:

fault the original source of any problem, such as a broken air pressure valve or assembly part out of tolerances, which *may* led to an *error* directly or indirectly.

error is a difference between what is specified and what is actually there. It may be latent (the system has not recognized it as such) or detected by a detection mechanism (often called monitor). Thus, in a control system, an *error* is an observable discrepancy between the actual state (the state in the controlled system) and the internal representation of the intended state.

failure occurs when an error affect the service delivered from a system in any way, e.g. the robot is unable to continue the grasping operation since someone have removed the object in question. Finally, (completing the chain of effects) if the component that experience a failure is a part of a system of components, the result is a fault in the system which contains the failing component.

Using this terminology, it is evident that what is commonly described as *error recovery* (such as catching a signal on division by zero) is really *failure recovery*, although it would be desirable to handle the problem before it manifests itself as a failure, i.e. to have true error recovery (detect the presence of a zero and do something about it before it is used in a division). Systems which are able to handle an error without affecting the service delivered are called *fault tolerant* systems.

The field of error recovery is often divided in three subfields:

detection techniques for (or the process of) observing the actual state of the controlled system and comparing it with specifications in order to find discrepancies as early as possible.

diagnose techniques for finding the original fault which caused the error.

recovery applying the proper corrective actions in order to prevent a possible future error or reach an error free state.

In each of these subfields there are several principles for how to achieve these objectives, as well as different methods for representing the information needed.

Classification

The following classification parameters will serve as a framework for the comparison of approaches and techniques. Although it is not always possible, subjective parameters has been avoided as far as possible. Thus no approach will be classified as being the 'best', since such an evaluation is dependent on the application and the motivation for using a specific approach in that application.

When presenting different approaches of error recovery, a basic categorisation is whether a *forward* or *backward* error recovery approach is used. The definition for these terms are ([Laprie, 1992; Noreils, 1990]):

backward error recovery: Find an earlier (previously passed) error free state of the system and return there, by 'undoing' what has been done since. Examples of general techniques using this approach are recovery blocks [Kim and Welch, 1989] and transactions [Harder and Reuter, 1983].

forward error recovery: Find an error free state that the system is supposed to eventually reach, and perform actions to reach that state. This is often done using predefined alternative actions or replanning of what actions to use to achieve the 'goal' [Chen and Trivedi, 1991; Noreils, 1990].

Other criteria which have implications on the system performance as well as its flexibility are:

Knowledge sources: The information (or knowledge) used may originate from several knowledge sources, such as from the original code, from a description of what to perform, from statistically aquired information about relations between error causes and effects in the system as well as the proper corrective action. This knowledge may also be manually specified after analysis of the system.

Application dependence: The knowledge as such may be more or less application dependent, which have implications for the reusability of the knowledge.

Knowledge representation: The knowledge used may be represented in several different *formalisms*, such as rules, alternative code or pieces of plans to use, graphs describing decision trees etc. Notable here is that several of these formalisms may be combined, or translated from one form into another.

Techniques for error recovery

In this section different aspects of error recovery will be presented and discussed. Firstly, techniques for the detection of a fault is discussed, followed by diagnose and recovery techniques.

Detection

In order to avoid costly failures, it is desirable to detect the presence of faults as early as possible. The constant verification of absence of errors is costly and not always possible, since the degree of observability of a system may not correspond to the richness in the model of the system. The following techniques address this problem.

Sensory signature is a term used by [Lee *et. al.*, 1983] to denote a collection of parameters which specify the limits of acceptability of a sensed signal during a specific task phase. The result is that sensor monitoring is guided by current action, not only in a binary fashion (should it be monitored or not) but also qualitatively (how tight are the limits, how often should they be verified).

Gini ([Gini, 1893]) uses a world model to represent the activity in a robot system. This world model and the operations on it is generated from the original robot program, and for each operation executed in the robot a corresponding semantic operation is executed in the world model. If the results differ (the world model containing the expected outcome compared to the sensed state of the world) there is an error. What sensors to use is guided by expected changes as well as expected lack of changes.

In [Hayes-Roth, 1990], Barbara Heyes-Roth uses a *focus of attention* principle in a life support and monitoring system to discriminate between which parameters to monitor more closely and not. This focus is guided by rules according to what task is performed, current load in the system, and may also (in case of overflow of sensor data) impose filtering strategies on sensor channels. The knowledge used is currently 'hard-wired', application dependent a priori knowledge.

Observations These three examples shows that there exists feasible techniques for detection, provided that there is a good observability of the system, that is that there are sensors enough to monitor most of the tasks. Notable is that the knowledge used to guide the monitoring in Gini's approach is extracted from the program, knowledge which the user must supply explicitly in the other two approaches. In Lee's approach, the knowledge is input twice (although in somewhat different forms) - both in the 'original' program and in the monitoring system.

Guiding the monitoring task in planning systems is often a variant of Gini's approach, and is thus not included here.

Omitted in this presentation is also techniques for how to implement monitoring of different types of signals, since this is out of the scope of this paper. However, an overview can be found in [Iserman, 1984] and [Frank, 1990].

Diagnose

Diagnose is the process of identifying the cause of the error (or failure). The identifcation starts by verifying the presence or absence of selected sensor data values, often called *features*. The result of the diagnosis is a set of *explanations* for the error, i.e. a set of possible faults.

Failure tree [Srinivas, 1977; Srinivas, 1978]: Early work by Srinivas presents a method where predefined knowledge about possible faults for each action (rules testing sensors and facts trying to explain why the action failed) and the current state of execution is used to build a *decision tree* of tests for 'features' upon a failure. This tree is then used to diagnose the situation. The method also uses the information available in the interrupted execution to trace situations where the error could be a result of a previous or parallel action. This is accomplished by examing the preconditions of all interrupted actions. Depending on the outcome of that examination, the corresponding error rules are also included.

Since Srinivas method is computationally expensive, it would be favourable to 'remember' the errors that actually occured in an application and their cause, and store them in a computationally more efficient form. The technique for doing so has been termed *machine learning*[1] [Chang and DiCesare, 1989; Chang *et. al.*, 1990a; Chang *et. al.*, 1990b][2] and is used to produce *heuristic rules* which given a failed action, tests for the presence and absence of certain features known to discriminate between possible explanations. The technique is based on an initial knowledge about possible errors for an action, and for each error both probable and definite features for that error. In the case when the system finds multiple explanations, it produces a tentative heuristic rule that may be confirmed or ruled out when more knowledge is acquired "from experiments, models or other sources".

Probability Vectors [Taylor and Taylor, 1988; Taylor *et. al.*, 1990] is an alternative technique where statistical a priori knowledge about the system is used. Each possible fault is associated with the probability of its occurence, and stored in a vector. This is repeated for all possible effects (errors), all model parameters and all sensors, resulting in four vectors containing probability estimations. Connection matrices describes a weighted relationship between faults and errors, errors and parameters, parameters and sensors. Using this information, it is possible to compute a set of probable causes for a given error, what sensors to use to verify an error as well as (at least in theory) guiding the process of instrumenting a work cell with sensors to achieve relevant observability.

Alternative approaches to diagnose can be found in other domains (such as AI in Medicine, e.g. see [Zhang, 1993]). They are often called *model based* or *consistency based*, since they contain a qualitative description of the system. This description, or model, expressed in some logical formalism, describes each components normal bahaviour and how modules interact. If an observation of an actual system state is added to the description such that the description becomes inconsistent, there is an error present. By identifying which component (or components) that in conjunction with the observation results in a contradiction, the set of explanations (faulty components) have been found. The expalnation is complete, the diagnose can handle multiple faults and may even reviel previously unknown faults [deKleer and Williams, 1987; Reiter, 1987]. However, it might sometimes propose awkward explanations since it lacks knowledge about possible errors, or rather, causal relations.

Adding *fault models* [deKleer and Williams, 1989; Struss and Dressler, 1990] to the description improves the explanation, but the explanation is no longer guarantied to be complete. The system may now label components as faulty, correct or either - meaning it can not decide which.

Observations The techniques for diagnose presented is just a small sample of what can be found in the literature. Characteristic for the first examples (failure trees with modifications and probability vectors) is that the knowledge is *shallow*, that is it is composed of 'rules of the thumb' or statistics about the system, and that it is not *complete*. If the system is reconfigured, large parts of the knowledge has to be updated. In the failure tree based approaches, this could be handled by specifying the diagnose knowledge separately for each module in an assembly cell. In a specific configuration, most of the knowledge would then be collected from its components, and only a small part would have to be added describing the current configuration. However, specifying all diagnostic knowledge for each module is probably not feasible, and thus they will still be incomplete - i.e. does not cover all failure situations.

In the model based approach, the knowledge stored is often called *deep* knowledge since it covers the normal structure and function of the system. This approach requires that the knowledge is complete, emphasing the same problems of uppdating the system as with the shallow techniques. Although this is an even more more computationally expensive approach, it has gained interest since it is better suited for modular description than the diagnostic rules above (e.g., see [Lee, 1986]).

Recovery

The term *(error) recovery* is used in a more versatile fashion than the two above. Essentially, it is the process of 'correcting an error', i.e. change the state of the controlled system to be consistent with specifications. If a module accomplishes this without failure (i.e. the system which uses this module/service never notices any error), the module is said to be *fault tolerant*. If the module reports the error to some other part of the system responsible for taking corrective actions, the system as a whole achieves *error recovery*.

1. Should not be mixed up with 'true' machine learning such as neural nets.

2. Extends and complements previous work such as [Chang *et. al.*, 1989] and [Pazzani, 1986; Pazzani, 1987].

The following sections reviews some of the approaches:

Programming language constructs A first step to structure the handling of exceptional situations in a programming language is to introduce a new programming construct, often called an *exception handler*. Upon an error, the control is transferred to the proper exception handler, which will then execute corrective actions. However, in some languages not even this support exists, and the constructs of the language has to be used to trap errors and accomplish dispatchers for corrective actions or default actions[1] [Cox, 1988; Cox 1989].

Attempts have been made to introduce a notion of *recovery points* (or blocks) in a program, specifying a legal and consistent state to return to in case of errors. This may be viewed as a weaker version of the concept of *transactions* as used in the data base community [Harder and Reuter, 1983], and there is ongoing work studying the usability of these concepts in the automation area [Schmidt, 1992].

In the data base community, the notion of SAGAS or nested SAGAS [Garcia-Molina and Salem, 1987; Garcia-Molina *et. al.*, 1991] has been developed and proposed for modelling parallel, nested activities in a corporation as a data base application. Examples of such activities are receiving orders, billing the customer while updating the inventory, and so on. In principle, it is a scheme for specifying compensating activities to be used in case of an abortion of an ongoing activity, and to specify how the abortion of one activity that is a part of a nested structure of activities should affect the other activities. There are no means to describe that some alternative activity should be performed when an activity aborts.

Knowledge based systems The term *knowledge based systems* is commonly used to denote any system where the knowledge used is separated from the program that uses it. These system are often also called *rule based* or *frame based* systems, according to how the knowledge is represented.

In two early proposals ([Lee *et. al.*, 1983; Gini, 1983]) a conventional robot system was extended by a "knowledge and reasoning module". Error recovery was achieved by using the failure tree produced in the diagnose process, and by augmenting the explanations with corrective actions/procedures. These 'corrections' were designed to restore the controlled system (the application) to either a previously visited state or a state further down the original program. In both these approaches the original robot program is interrupted and the recovery is performed by downloading new instructions to the robot.

In Noreils *et. al.* [Noreils and Chatila, 1989; Noreils, 1990] proposes a variant of the solution above. Here the corrective action is inserted into the original program, either replacing the original (failed) part or as a corrective continuation of it, and the program execution is then continued by the interpreter in its current context. This approach is often called *plan repair* - and the idea is to 'remember' the correction by altering the plan/program.

Delchambre *et. al.* presents a backward error recovery approach [Delchambre and Coupetz, 1988; Gaspart *et. al.*, 1989]. It uses a 'flat' (non hierarchical) plan representation with framelike structures representing parts and their features, and rules describing general assembly knowledge. The plan describes the assembly task and is given by the user. An example of a feature defined for a part is the 'handability coefficient', describing how 'easy' it is for the robot to handle that part. An example of an assembly knowledge rule is that if a sub-assembly is to be picked up, the system should focus the picking operation to the part of the sub-assembly with the best handability coefficient. The response to an error is to disassemble the faulty part (i.e. only errors caused by faulty parts are handled) by generating a sequence of actions (a plan) to accomplish this.

Meijer *et. al.* presents a hierarchical system in which the objects used in the (a priori given) plan is called 'knowledge areas' (KA's) [Meijer and Herzberger, 1988; Meijer *et. al.*, 1991]. Each KA is responsible for some action such as calling a partially ordered set of other KA's or performing some primitive sensing or acting. For each KA there is an invocation specification, specifying what goals it will fulfill and/or what facts need to exists in order to use it. If it is a primitive KA, monitor conditions may be specified, as well as a set of exception handling strategies. Since these strategies are themselves represented as KA's, they contain invocation specifications guiding the system to what recovery strategy to use when the monitoring conditions signal an error.

A hierarchical, frame based approach is proposed by [Chen and Trivedi, 1991], where frames represent plan skeletons of different abstraction levels. The 'planning' consists of selecting appropriate 'frames' and 'instantiating' subplans (frames) refining the plan until all leaves are primitive actions or perceptions. In this approach, the major difference between generating the original plan for an assembly task and generating an error recovery plan is that the overall goal for the plan is only considered in the first case. During planning for error recovery, the main goal is to achieve any error free state.

The main differences in the approaches taken by Meijer and Chen is that in the first case exception handling strategies are predefined for each primitive action, albeit as more or less general plans, while in Chen's approach these plans are generated upon an error situation using available actions and subplans, but with a somewhat different goal state than during the original planning.

Graph based approaches When the task description is based on some graph formalism such as a Petri Net or Finite

1. Examples are 'longjump' in C and structured use of methods in C++ .

State Machines, some restricted variants of the approaches described above apply. In the following, only Petri Net based approaches will be presented since alternative formalisms are often compilable into a Petri Net.

Petri Nets (PN) is a static description (based on *places* and *transitions* between them, where action normally take place during a transition) without any exception handling constructs as can be found in programming languages. Thus, if error handling is to be used in a PN based approach, it must be modelled as a part of the normal operation, and once the PN is constructed it can not be arbitrarily modified. Using PN's, there are four constructs that can be used to encode exception handling [Zhou and DiCesare, 1989]:

Input Condition: When several transitions leaves a place, it is customary to select arbitrarily between them. However, the transitions may be augmented with a condition, which if satisfied will favour the selection of that transition, and otherwise reject that transition.

Alternative Path: Using the Input Condition construct above, an alternative path through the PN may be defined.

Backward Error Recovery: The two constructs above is used to trap an anomaly and execute corrective actions (undoing the problem), returning to a previously visited place in the PN.

Forward Error Recovery: Analogous to Backward Error Recovery, but the corrective action will directly solve the problem and return control to the same place as where the error originated (or was detected).

Petri Nets with backtracking: In [Cao and Sanderson, 1992] a system is presented which generates a PN controller from an assembly description represented as an AND/OR tree. At each place reached by a transition where the transition/operation may fail, a forward error recovery action is defined which will try to correct the failure. If the PN still fails, a backtracking approach is used based on the augmentation of the PN with inverse transitions (brother transition in [Cao and Sanderson, 1992]) allowing the system to backtrack (disassembling parts) to a previous state. In the assembly context, depending of the parts disassembled, the system may choose to redo the assemble operation with partially new material as a strategy to correct the original error. This augmentation of the PN is done as a part of the generation of the PN from the AND/OR graph, and can only handle a subset of all faults that may occur in an assembly cell. Also, when expanding the PN with new places and transitions for handling anomalies, the resulting net grows large and complex.

Layered Petri Nets (LPN) is used to modularize a PN controller as several PN's responsible for different functionality or modes of operation. In a LPN a place may be defined to be a complete PN in itself, defining a complex action to be performed when the token reaches that place. In [Hasegawa *et. al.*, 1990] LPN's is used to define separate nets for normal operation (an auto-mode net) for exception handling, manual operation etc., and how to switch between these modes. This may be viewed as a more general and structured version of the exception handler construct mentioned above, since it also handles 'manual mode' and any other mode of operation desirable.

Modifiable Layered Petri Nets: In [Zhou and DiCesare, 1989; Zhou and DiCesare, 1993] a LPN[1] using the four error recovery methods described above in cooperation with an Error Diagnose and Recovery Planning module (EDRP) is presented. When an anomaly appears during 'execution' of the LPN which has previously not been planned for, the EDPR module extends the original LPN by transforming parts of it (using the four constructs described above) to handle the situation. This may be viewed as on line 'patching' of the code, but the authors claim that all properties of the original net (such as avoiding deadlocks or buffer overflows) is preserved[2].

Observations

Table 1: Techniques for recovery after an error

	Type of error recover	Knowled ge source?	Is knowl-edge appli-cation dependent	Knowled ge repre-sentationt echnique
Gini	any	program	yes	-
Lee	any	user	yes	rules
Noreils, Chatila	any	user/system	yes/no	rules
Delchambre	backward	user/system	no	rules,frames
Meijer *et.al.*	any	user	?	sets of rules
Chen	any	system	no	plans
Cao *et. al.*	backward	assembly	yes	petrinet
Hasegawa *et. al.*	any	user	yes	layered PN
Zhou, DiCesare	any	user/system	yes/no	layered PN, rules

Most of the techniques presented above provide both back-

1. Here the authors views the *place* as an operation or ongoing activity and a *transition* as an instantaneous change of state (place).

2. Theoretical work supporting this is described in [Fielding *et. al.*, 1988].

ward and forward error recovery. Some of the systems combine predefined system information with application dependent knowledge specified by the user (Delchambre/ Zhou), some extract it from other available sources such as the original robot program or assembly description(Gini/ Cao). Notable is that very few approaches uses general error recovery knowledge only. Exceptions is approaches proposed by Delchambre and Chen, witch manages without any application specific error recovery knowledge. In all other approaches error recovery knowledge is associated with each action or each possible error explanation.

Techniques for recovery seems to evolve in two directions; techniques based on graph formalisms such as Petri Nets, which have some tractable properties of provable liveliness etc., and general, plan based formalisms. In both cases there are planners involved, which poses a problem since most planners are known to have problem with soundness[1] and/or safeness[2], and to have intractable complexity (i.e. they can't handle large data). However, there are planners that are safe, and in some domains even sound. And by restricting the semantics of the planning problem, the complexity can be reduced. Promising work in this direction can be found in [Klein, 1993].

Conclusions

In this paper different techniques proposing solutions to the error recovery field have been reviewed. What they share is a completely different view of what to specify and how to do it compared with equipment traditionally used in industry, which makes them hard to integrate into existing production plants. Although, some attempts have been made to build industrial applications (e.g. Thorn EMI [Ashton *et. al.*, 1987], and Lookheed Aeronautical Systems Company [Kartak, 1988]), using only the most fundamental techniques. As limited as they are, they are still regarded as improvements to the alternatives given at the time.

As striving towards a more flexible work cell equipment yields a more complex instruction task and a more complex behaviour, it is essential to find suitable high-level methods and languages for instructing the cell if the flexibility is not to be lost. Simply providing 'hi-level' languages used for programming computers is not enough. The goal must be to maximize the expressiveness while minimizing the specification needed, and thus only leave to the application programmer to specify what is really application dependent knowledge, knowledge which can not be generated from the design of the product. This in turn implies that different parts of the system (abstraction levels) is to be specified/instructed by different categories of people, and thus that different languages/forms av descriptions may be needed - as opposite to what is suggested in [Cox, 1988].

Concluding from this, future research in the area ought to be heading towards multileyered knowledge based systems describing limited domains, systems which is easy to adapt to a new application.

1. Just because the planner does not find a plan there is no guarantee that it doesn't exist one

2. The planner is not guaranteed to halt if a plan does not exist

References

[Ashton et. al., 1987] M. Ashton, D. A. Harding and M. I. Micklefield. A flexible assembly system controller. In *Proceedings of the 2nd International Conference on Machine Control Systems - MACON-2*, p.165-74, IFS Publications, Kempston, UK 1987.

[Cao and Sanderson, 1992] T. Cao and A. C. Sanderson. Sensor-based Error Recovery for Robotic Task Sequences Using Fuzzy Petri Nets. In *IEEE International Conference on Robotics and Automation*, p.1063-9, 1992.

[Chang and DiCesare, 1989] S. J. Chang and F. DiCesare. The generation of diagnostic heuristics for automated error recovery in manufacturing workstations. In *IEEE International Conference on Robotics and Automation.* p. 522-7 vol.1, 1989.

[Chang, 1989] S. J. Chang, F. DiCesare and G. Goldbogen. An algorithm for constructing a failure propagation tree in manufacturing systems. In *IEEE International Symposium on Intelligent Control*, p.38-43, 1989.

[Chang et. al., 1990a] S. J. Chang, G. Goldbogen and F. DiCesare. Evaluation of diagnosability of failure knowledge in manufacturing systems. In *IEEE International Conference on Robotics and Automation*, p.696-701 vol.1, 1990.

[Chang et. al., 1990b] S. J. Chang, G. Goldbogen and F. DiCesare. Aspects of diagnostic rules for manufacturing systems: generation, generalization and reduction. In *IEEE International Conference on Systems, Man and Cybernetics*, p.78-83, 1990.

[Chen and Trivedi, 1991] C. X. Chen and M. M. Trivedi. A task planner for sensor-based inspection and manipulation robots. In *Proceedings of the SPIE - The International Society for Optical Engineering*, vol.1571, p. 591-603, 1991.

[Cox, 1988] I. J. Cox. C++ language support for guaranteed initialization, safe termination and error recovery in robotics. In *IEEE International Conference on Robotics and Automation.* p.641-3 vol.1, 1988.

[Cox, 1989] I. J. Cox and N.H. Gehani. Exception handling in robotics. *Computer*, 22(3):43-9, March 1989.

[deKleer amd Williams, 1987] J. deKleer and B.C. Williams. Diagnosing Multiple Faults. *Artificial Intelligence*, 32(1):97-130, 1987.

[deKleer and Williams, 1989] J. deKleer and B.C. Williams. Diagnosis with Behavioural Modes. In *Proceedings of the 1989'th Intenational Joint Conference on Artificial Intelligence* (IJCAI89), pp.1324-1330, 1989.

[Delchambre and Coupez, 1988] A. Delchambre and D. Coupez. Knowledge based error recovery in robotized assembly. In *Proceedings of the 9th International Conference on Developments in Assembly Automation - Japan vs Europe; Product Design for Assembly; Assembly Automation.* p.349-66. IFS Publications, Kempston, Bedford, UK, 1988.

[Fielding *et. al.*, 1988] P. J. Fielding, F. DiCesare and G. Golbogen. Error Recovery in Automated Manufactoring through the Augmentation of Programmed Processes. In *Journal of Robotic Systems*, 5(4), 337-362, 1988.

[Frank, 1990] P. M. Frank. Fault Diagnosis in Dynamic Systems using Analytical and Knowledge Based Redundancy - A Survey on some new results. *Automatica*, 26(3):459-474, 1990.

[Garcia-Molina and Salem, 1987] H. Garcia-Molina and K. Salem. SAGAS. Proc. SIGMOD int. conf. on Management of Data, pp.249-259, May 1987.

[Garcia-Molina *et. al.*, 1991] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner and Kenneth Salem. Coordinating Multi-Transaction Activities. *Data Engineering Bulletine*, 1991.

[Gaspart *et. al.*, 1989] P. Gaspart, A. Delchambre, A. Coupez and P. Brouillard. Rule based procedures for diagnosis and error recovery. In *Proceedings of MIV-89 - International Workshop on Industrial Applications of Machine Intelligence and Vision* (Seiken Symposium), p.88-93, 1989.

[Gini, 1983] M. Gini. Recovering from Failures: A New Chalenge for Industrial Robotics. In *Proceedings of the 25'th IEEE Computer Society International Conference (COMPCON-83)*. p.220-227, Arlington 1983.

[Harder and Reuter, 1983] T. Harder and A. Reuter. Principles of Transaction Oriented Database Recovery. *ACM Computing serveys*, 15(4):287-317, 1983.

[Hasegawa *et. al.*, 1990] M. Hasegawa, M. Takata, T. Temmyo and H. Matsuka. Modelling of exception handling in manufacturing cell control and its application to PLC programming. In *IEEE International Conference on Robotics and Automation*, p.514-19, vol.1, 1990.

[Heyes-Roth, 1990] B. Hayes-Roth. Architectural Foundations for Real-Time Performance in Intelligent Agents. In *Journal of Real-Time Systems*, no.2, p.99-125, 1990.

[Iseremann, 1984] R. Isermann. Process Fault Detection based on Moddeling and Estimation Methods - A Survey. Automatica, p.387-404, vol.20, 1984.

[Kartak, 1988] J. A. Kartak. Development of automated workcell control software: a case study. In *Proceedings of the 18th International Symposium on Industrial Robots*, p.467-91, 1988.

[Kim and Welch, 1989] K. H. Kim and O. H. Welch. Distributed Execution of Recovery Blocks: an approach for uniform treatment of hardware and software faults in real-time applications. In *IEEE transactions on Computers*, 38(5):626-636, 1989.

[Klein, 1993] I. Klein. Automatic Synthesis fo Sequential Control Schemes. PhD-theses no.305, Linköping University, 1993.

[Laprie, 1992] J. C. Laprie. Basic Concepts and Associated Terminology. In *Dependable Computing and Fault Tolerant Systems*, Vol. 5, Springer-Verlag, Wien New-York, 1992.

[Lee *et. al.*, 1983] M. H. Lee, D.P.Barnes and N.W. Hardy. Knowledge Based Error Recovery in Industrial Robots. In *International Joint Conference on Artificial Intelligence* (IJCAI83). pp.824-26, 1983.

[Lee, 1986] M. H. Lee. Deep knowledge modelling in robotics. In *Proceedings of the Alvey IKBS Research Theme: Expert Systems. Deep Knowledge. Workshop No.2*, p.44-50, Alvey Directorate, London, UK, 1986.

[Meijer and Herzberger, 1988] G. R. Meijer and L. O. Hertzberger. Off-Line Programming of Exception Handling Strategies. In *Proceedings of IFAC Symposium on Robot Control*. p.431-436, Karlsruhe 1988.

[Meijer *et. al.*, 1991] G. R. Meijer, L. O. Hertzberger, T. L. Mai, E. Gaussens and F. Arlabosse. Exception Handling System for Autonomous Robots Based on PES. In *Journal of Robotics and Autonomous Systems*. 7(2-3):197-209, 1991.

[Noreils, 1990] F. R. Noreils. Integrating error recovery in a mobile robot control system. In *IEEE International Conference on Robotics and Automation*, p.396-401 vol.1, 1990.

[Noreils and Chatila, 1989] F. R. Noreils and R. G. Chatila. Control of mobile robot actions. In *IEEE International Conference on Robotics and Automation*. p. 701-7 vol.2, 1989.

[Pazzani, 1986] M. J. Pazzani. Refining the Knowledge Base of a Diagnostic Expert System: An Application of Failure Driven Learning. In *5'th National Conference on Artifical Intelligence (AAAI-86)*. p.1029-35, 11-15 Aug. 1986.

[Pazzani, 1987] M. J. Pazzani. Failure-Driven Learning of Fault Diagnosis Hheuristics. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-17(3), p.380-394, May/June 1987.

[Reiter, 1987] R. Reiter. A Theory of Diagnose from First Principles. *Artificial Intelligence*, 32(1):57-95, 1987.

[Schmidt, 1992] U. Schmidt. A Framework for Automated Error Recovery in FMS. In *International Conference on Automation, Robotics and Computer Vision*. p. IA.3.4.1-5, 1992.

[Srinivas, 1977] S. Srinivas. Error Recovery in Robot Systems PhD thesis California Inst. of Tech., Pasadena, 1977.

[Srinivas, 1978] S. Srinivas. Error Recovery in Robots Through Failure Reasoning Analysis. In *Proceedings of AFIP - National Computer Conference*, p.275-282, 1978.

[Struss and Dressler, 1989] P. Struss and O. Dressler. "Physical Negation" - Integrating Fault Models into the General Diagnostic Engine. In *International Joint Conference on Artificial Intelligence* (IJCAI89), pp. 1318-1324, 1989.

[Taylor and Taylor, 1988] G. E. Taylor and P. M. Taylor. Dynamic error probability vectors: a framework for sensory decision making. In *IEEE International Conference on Robotics and Automation*, p.1096-100, 1988.

[Taylor *et. al.*, 1990] P. M. Taylor, I. Halleron and X.K. Song. The application of a dynamic error framework to robotic assembly. *IEEE International Conference on Robotics and Automation*, pp170-5, 1990.

[Zhang, 1993] T. Zhang. A Study in Diagnosis Using Classification and Defaults. PhD thesis no. 302, Linköping University, 1993

[Zhou and DiCesare, 1989] M. C. Zhou and F. DiCesare. Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19(5): 963-73, Sept. 1989.

[Zhou and DiCesare, 1993] M. C. Zhou and F. DiCesare. Petri Net Synthesis for Discrete Event Control of Manufactoring Systems. ISBN 0-7923-9289-2, Kluwer Academic Pub., 1993.