Preprints of IAD' 2007: IFAC WORKSHOP ON
INTELLIGENT ASSEMBLY AND DISASSEMBLY
May 23-25 2007, Alicante, Spain

FAILURE RECOVERY PLANNING FOR ROBOTIZED ASSEMBLY BASED ON LEARNED SEMANTIC STRUCTURES

Luís Seabra Lopes

Transverse Activity on Intelligent Robotics
IEETA/DETI – Universidade de Aveiro
3810-193 Aveiro, Portugal

Abstract: In complex task domains such as assembly and disassembly, robots need to reason about the tasks and the environment in order to make decisions. Reasoning should be based on experience acquired by the robot through interaction with users and other robots. This paper focuses on the use of learned semantic structures for failure recovery in assembly, a classical problem that, nevertheless, is far from receiving from the robotics community the due attention. A method for failure recovery planning guided by learned semantic knowledge is described, formally analyzed and empirically evaluated.

Keywords: Intelligent manufacturing systems, assembly, robotics, system failure and recovery, artificial intelligence, planning, machine learning

1. INTRODUCTION

In flexible manufacturing and assembly systems (FMS/FAS), the keyword *flexibility* is generally understood as *the ability to cope with change*. This is a particularly important topic, especially in what concerns failure detection, diagnosis and recovery. In the assembly area, if other flexibility elements, such as sophisticated end-effectors and flexible, especially modular fixtures, are introduced, then robots will potentially be able to perform a greater variety of tasks with a higher complexity. This implies that the robot should be able to choose which actions to perform in each situation.

My main research interests are concerned with the development of integrated robot architectures that support reasoning and learning at the task level (Seabra Lopes and Connell, 2001). In this paper, I focus on work concerned with failure recovery for robotized assembly. Failure recovery and related problems must be addressed at multiple levels, including prevention, fault tolerance, action-level failure recovery, task-level failure recovery, reactive and deliberative issues, human-robot interaction issues, etc.

The failure recovery problem still needs a lot of research from the Artificial Intelligence, Robotics and Automation communities Literature review (Loborg, 1994; Toguyéni *et al*. 2003; Zielinski, 2002) shows that only a very small number of papers on failure recovery are published each year in the main journals and conferences. A distinction is often made between local (action-level) and global (task-level) recovery (Lopez-Mellado Alami, 1990). Local recovery is often handled through pre-defined procedures (Loborg and Törne, 1994; Lopez-Mellado Alami, 1990; Meijer, 1991). When such procedures fail, typically the human operator is called.

For global recovery, López-Mellado and Alami (1990) propose to execute any actions that, according to the partially ordered task plan, do not depend on the failed action. Meanwhile, the human operator is called to solve the detected failure. Only in very few cases, has automated planning been used for failure recovery. Meijer (1991), while proposing a similar "task re-scheduling", additionally proposes the use of a limited-horizon planner to try to devise a failure recovery plan automatically. Only when that is not possible will the human operator be called. Evans

and Lee (1994) use planning for failure recovery, but the generated plans are still simple enough to be converted to reactive procedures. After these efforts in the early 1990's, very little work has been concerned with failure recovery planning.

There are several, complementary, approaches for tackling the failure recovery planning problem:

1. A search-based planner is able to come up with a failure recovery strategy.
2. The robot has enough time to go through trials and eventually comes up with a solution; this is a kind of search, but carried out in the physical world (Ferch and Zhang, 2001).
3. The robot has been through a similar problem before and was able to solve it; the strategy that was used can now be adapted.
4. If the above approaches fail, an external agent (e.g. a human teacher) can provide an appropriate recovery strategy.

The major problem faced by artificial intelligence planners in the domain of failure recovery is concerned with the search complexity involved in real-world planning. A robot has limited time to plan and recover from a situation. Even in very circumscribed domains, planning complexity prevents planners from finding solutions to problems that require long action sequences. In real-world domains, of course, the number of action alternatives is much bigger and complexity becomes a problem, even when only a short sequence of actions is to be determined. For instance, in the assembly domain, a sophisticated hand or a modular fixture offer a variety of possibilities for solving a variety of tasks. However, finding the right sequence of arm moves and hand and fixture configurations for solving an unexpected problem is not easy. Heuristics are often used to focus search. The problem is that it is extremely difficult to define heuristics for the domain of failure recovery planning. Going through trials in a framework of reinforcement learning is something that the robot can do in spare time or in a training period. Work of this type in the domain of cooperative grasping and assembly is reported in (Ferch and Zhang, 2001). However, this does not seem viable for failure recovery in normal operation.

Avoiding planning complexity by using knowledge about solutions to similar, previously encountered problems is a way out that has been attracting increasing attention from researchers in various domains. I have developed a failure recovery approach that involves recognizing the basic principles underlying solutions to concrete cases and the application of those principles to new situations. (Seabra Lopes, 1999). In the failure recovery domain, the descriptions of failure categories and operator schemata make up a domain theory that must be taken into account when explaining (or understanding) the recovery strategy that was applied in a given situation. Learning the basic principles of the applied solutions involves a series of deductive and inductive inference steps. In recovery planning, the inverse transformations are applied.

In this paper, I take up previous work by providing formal definitions, formal complexity analysis of the planning algorithm and experimental results. For the learning algorithm, refer to (Seabra Lopes, 1999). This work takes the common assumption in AI planning, namely that the world normally changes as described in action models. In case of failure, this work also assumes that failure analysis capabilities will enable to update the world model and start the recovery planning activity. The work finally assumes that the manipulated representations are perceptually grounded.

2. FAILURE RECOVERY KNOWLEDGE

A symbolic and logic-based approach is adopted. The world model is a set of logic formulas that represent, not only permanent information about the world, but also the momentary situation. The world can change through actions of the robot and unexpected events. The types of actions the robot is able to perform are represented by PDDL type operator schemata (Gerevini and Long, 2005). An operator schema specifies the pre-conditions and effects of an action. Plans are obtained by stringing together operator applications, i.e. instanciated operator schemata.

The execution of a plan can fail to deliver its goals. An *execution failure* is, therefore, a deviation between the expected and the actual situation of the world detected during plan execution. Failure detection requires a functionality usually called *execution monitoring* (Pettersson, 2005). Execution failures can be caused by *system faults* (malfunctions in equipment), *external exceptions* (e.g. some unexpected occurrence in the robot environment) or previous execution failures (Camarinha-Matos *et al*., 1996). An execution failure is characterized by a certain number of effects on the situation of the world. Thus, the proposed representation resembles that of operators:

Definition 1 - A *failure category* describes a class of execution failures in terms of the execution context and failure effects. It is represented as a tuple $FC = \langle FT, OT, DL, AL \rangle$, where:

- *FT* is a functional expression representing the *failure category template*.
- *OT* is the *failed operation template*. (In general, this could be some pointer to the *execution context*)
- *DL*, called the *delete list*, is a list of atomic (logic) formulas that should be deleted from the description of the situation of the world prior to executing the failed operation.
- *AL*, called the *add list*, is a list of atomic formulas that should be added to the situation description.

The description of a failure actually occurred in the physical scenario is an instanciation of a failure category and is identified by the instanciated failure

template. Determining failure effects after failure detection can be done through a failure diagnosis module, which should be part of the agent architecture. Failure diagnosis includes failure *confirmation*, *classification* and *explanation*, and concludes with *status identification*. Based on the results of failure diagnosis, a failure recovery strategy can be devised.

Applying a failure recovery strategy is a learning opportunity, whether it is successful or not. Success means, of course, that the planned recovery operations are completely executed, allowing to resume execution of the nominal plan. If some exception, unrelated to the initial failure, causes a new failure which is handled recursively, and then the initial recovery strategy is resumed and completed, recovery is also considered successful. Unsuccessful recovery may happen due either to the application of an incorrect recovery strategy, given the initial failure diagnosis, or to errors in the initial diagnosis itself. The first case can usually be avoided by verifying the effects of the recovery strategy, according to the applied operator schemata, before execution. In the second case, learning is more difficult because it involves criticizing and reformulating the used diagnostic model. In this paper, learning is attempted only from successful failure recovery episodes.

The following representation is adopted:

Definition 2 - A *failure recovery episode* is a tuple <*OT*,*FT*,*RP*>, where *OT* is the *failed operation template*, *FT* is the *failure instance template* and *RP* is a plan (i.e. a sequence of operator applications) used to recover from the failure (*recovery plan*).

The basic principles that explain the success of a failure recovery strategy are then extracted based on several deductive as well as inductive transformations, namely deductive generalization, abstraction, feature extraction and clustering of repeated plan patterns (Seabra Lopes, 1999). The final result of these learning computations is a failure recovery schema, which can now be formally defined:

Definition 3 - A *failure recovery schema* describes guidelines for building solutions for instances of a given failure category. It consists of a tuple <*OT*,*FT*,*RSK*>, where:

- *OT* is the template of the applied operator.
- *FT* is the failure category template.
- *RSK* is the recovery plan skeleton, consisting of a sequence of abstract steps described as pairs <*Action*,*Features*>, where *Action* is either an abstract operator or a sequence of abstract operators and *Features* is a list of features of the objects manipulated by *Action*.

Each abstract operator can be concretized through a sequence of task-level operators. Failure recovery schemata can be thought of as conceptual categories, but not in the usual sense of the term. Although these categories organize similar concepts, their members cannot be enumerated. They can, however, be reconstructed as necessary.

An example of a failure recovery schema, learned with this approach from a 24-steps plan in the assembly domain described in (Seabra Lopes, 1999), is presented in figure 1.

```
[ assemble(R,T,Obj1,Comp1,Type1,Prod,Fix),
  defective_assembly(DefObj,Comp4,Type4,Prod,Fx),
  [ [ place(Obj1), [ ] ]
    [ [disassemble(X,C), place(X)],
         [ initially_assembled(C),
           assemble_after(C,Comp4),
           assemble_before(C,Comp1) ] ]
    [disassemble(DefectiveObj, Comp4), [ ] ]
    [place(DefectiveObj), [ ] ]
    [pick(NewObj),
         [ same_type_as(NewObj,DefectiveObj) ] ]
    [assemble(NewObj,Comp4),
         [ same_type_as(NewObj, DefectiveObj) ] ]
    [ [pick(X), assemble(X,C) ],
         [ initially_assembled(C),
           assemble_after(C,Comp4),
           assemble_before(C,Comp1)]]
    [pick(Obj1), [ ] ]
    [assemble(Obj1,Comp1), [ ] ]
]  ]
```

Fig. 1. Example of learned failure recovery schema (see Definition 3.)

It covers a range of assembly failure situations caused by a defect in an assembled component and the subsequent impossibility of assembling another component. The learned strategy contains two clusters, one for the disassembly phase, `[disassemble(X,C), place(X)]`, and the other for the assembly phase `[pick(X), assemble(X,C)]`, where variables `X` and `C` are constrained by certain features (given in italic). For instance, feature `assemble_before(C,Comp1)` specifies that assembly component `C` must be assembled before `Comp1`, i.e. before the component whose assembly failed.

3. PLANNING GUIDED BY AN ABSTRACT PLAN

When a failure is detected and the diagnosis function is able to produce the failure description, recovery planning is attempted. The first step is to look for similar failure situations previously encountered. It shouldn't be ignored that a so-called "utility problem" has been identified for learning systems, particularly for explanation-based learning systems (Minton, 1990). It arises when the added costs of matching learned control rules exceed the search guidance benefits. Similar problems can occur in case-based systems, which is based on storing cases. In the proposed approach, the combination of deductive generalization with different forms of abstraction (including clustering) enables to learn failure recovery schemata that cover a wide range of situations, therefore reducing the utility problem considerably. If needed, indexing/retrieval techniques

developed within CBR (Kolodner, 1993) should be used.

In the work described here, the failed operation template plus the failure instance template are used as indexing key to locate and retrieve a suitable a failure recovery schema. Then, the plan skeleton in that schema is concretized to solve the current situation.

3.1 Planning Guided by an Abstract Plan

Concretizing a plan skeleton, i.e. adapting it such that it solves a particular situation, can be done by planning a solution for the situation and using the skeleton simply to guide the search. If the plan skeleton contains no clusters, an A* graph-based progression planning algorithm can be directly applied. In this case, all that must be done is to redefine the cost function to take into account the information contained in the plan skeleton. The previously extracted features play an important role in cost computations.

The following cost estimation method was implemented:

- At the root node of the search space, representing the failure situation, the entire plan skeleton is considered as the guide to reach a situation in which the failure is considered recovered. In each node of the search space, there is some remaining part of the initial plan skeleton that provides guidance for going from that node to a solution node.
- When expanding a node, the first abstract operation in the remaining part of the plan skeleton, is used to judge the utility of different operator applications and assign them different costs. Let *OP* be a legal operator application in the current node and let C_R be the real cost of executing it [1]. Let *AO* be the head of (i.e. the first abstract operation in) the current plan skeleton. The cost of *OP* actually considered by the planner, C_P, is defined by the following rules:
  - If *OP* is an auxiliary operation, therefore belonging to the *nil* class in the operator abstraction hierarchy, then C_P will be the real cost C_R.
  - If *OP* belongs to the category of *AO*, then let K be the number of features documenting *AO* in the plan skeleton and let $V \leq K$ be the number of these features actually verified (i.e. true) in the case of *OP*. In this case, we have
    $$C_P = C_R\,(K+1)/(V+1)$$
    This way, operator applications that verify a greater number of features of the abstract operation will be preferred. If $V = K$, then C_P will be the real cost.
  - In all other cases, C_P will be infinity. In this case, the planner won't even create the corresponding child node in the search tree.
- When a node n is created, its total cost is defined as $f(n) = g(n) + h(n)$, where: $g(n)$ is the sum of the C_P costs of all operations in the path leading from the root node to n; the estimated cost of reaching a solution node is $h(n) = \alpha \cdot L$, where L is the length of the current abstract plan and α is an estimate of the average cost of operations in a task-level plan per each operation in the abstract plan [2].

3.2 Expanding Plan Clusters

The planning process becomes more complex when the plan skeleton contains clusters, due to the uncertainty with respect to the number of cluster instances that are required for the particular problem. Since this problem interacts with the structure of the entire plan skeleton, the best approach is to carry out an abstract planning phase, in which the clusters in the plan skeleton are replaced by sequences of abstract operations and the parameters of the abstract operations are instanciated. A greedy progression planning approach is followed in converting a plan skeleton with clusters into a mere sequence of abstract operator applications. When the planner encounters a cluster, it can either skip it or append a cluster instance to the current plan skeleton.

In this approach, cost is not taken into account. Only a measure of preference of a particular abstract operator application over alternative applications is used. Let *AO* be an abstract operation in the plan skeleton and let K be the number of features that documents it. Let *AA* be a possible application of *AO* and let V be the number of features in *AO* verified in *AA*. *AA* will be preferred over other operator applications if it has the highest value of the ratio $(V+1)/(K+1)$. Note that, as a direct consequence of taking into account features in cost computations, recovery planning will be easier for those failure episodes that are more similar to the failure episode that originally enabled learning. Precisely the same occurs with human learning.

4. ANALYSIS AND EVALUATION

4.1 Computational complexity of planning

Planning from experience, following the method described above, assumes that the selected failure recovery schema is indeed applicable to the situation being addressed. As already mentioned, such analysis is to be carried out by a diagnosis module. If the schema is not applicable to the situation, planning

[1] For tests in the assembly domain reported below, all costs were set to 1.0.

[2] For tests in the assembly domain reported below, $\alpha = 2.5$ was used.

will eventually fail. This happens because, in order to focus search, the plan skeleton is used to actually cut off a large portion of the search space.

When planning from scratch, the average branching factor is

$$B = N_{ao} \times B_{ao} + B_{nil}$$

where N_{ao} is the number of abstract operators defined for the domain, B_{ao} is the average number of instantiations of an abstract operator applicable on any given state and B_{nil} is the average number of auxiliary operations applicable on any given state. In the particular domain used for the examples of this paper, B has been experimentally determined to be around 10 operations (Seabra Lopes, 1999).

When planning from experience, according to the algorithm presented in section 3.1, the average branching factor is reduced to $B = B_{nil} + B_{ao}$. If, for instance, we have a domain in which $N_{ao} = 4$ and $B_{nil} = B_{ao}$, we see that the branching factor is reduced to 40% of the original branching factor. This leads to a considerable reduction in the search space. For larger domains, this improvement will be even more significant. However, it does not prevent an exponential growth in the general case.

The actual performance of the proposed planning approach in a particular situation depends on the following two main conditions:

a) The current situation is similar enough to the situation that originally enabled learning, so that the same basic solution can be applied, without any extra search effort;
b) The features used to document each step of the abstract plan skeleton are sufficiently informative, therefore allowing an appropriate similarity assessment.

Situations meeting these two conditions are the main target of the proposed planning approach. Under these conditions, the approach has been experimentally observed to exhibit a linear computational complexity (see section 4.2). More specifically, the number of created nodes in the search tree grows linearly with the size of the solution plan, resulting in a linear complexity both in terms of memory usage and running time. This can also be intuitively seen through an analysis of the planning process. Typically, since the abstract plan skeleton doesn't provide any guidance with respect to auxiliary operations, most of the search effort is consumed in finding appropriate auxiliary operations to insert between the other task-level operations. The search for intermediate auxiliary operations terminates as soon as a state is generated where an appropriate instance of the next abstract operation can be applied.

Assuming that the features characterizing the abstract operation are informative and, therefore, the right task-level operation is selected in the first attempt, the average depth of the local search for auxiliary operations is given by $l = 1 + \lambda/(1-\lambda)$, where λ is the proportion of auxiliary operations in the solution plan. In these circumstances, the size of the local search space, S, grows exponentially [3] with l, but can be considered independent of the length of the full solution plan. The number of expanded nodes for a full recovery plan of size L will be in the order of $(1-\lambda) \times L \times S$ which is linear in L. When features are not enough to pinpoint the right task-level operation to implement a given abstract operation, the local search space will tend to expand beyond that point. The more often that happens, the less linear the search process will be.

4.2 Empirical evaluation

Empirical evaluation was carried out in the assembly domain described in (Seabra Lopes, 1999). Parts are available to the assembly robot through feeders or in pallets and tools are picked up from a tool magazine. The product used in the examples is the Cranfield Benchmark, a classical pendulum-like structure used for evaluation of research in the assembly domain (Rathmill and Collins, 1984). 13 different action categories (planning operators) are used. An assembly plan for the Cranfield benchmark, considering assembly operations and auxiliary operations for part feeding and fetching, has a length of around 50 operations. The average branching factor of the planning search space has been observed to be approximately 10. This means that, if there are no heuristics for guiding the search, a search space of 10^n may have to be explored for determining a plan with n operations.

For this evaluation, the failure recovery schema presented in fig. 1 was used to solve five recovery planning problems (Table 1). First of all, the learned schema was used to solve the problem based on which it was learned. An optimal solution, equivalent to the one used for learning, is obtained, after expanding a search tree with an Average Branching Factor of 4.299 and an Effective Branching Factor of 1.197 (first line in Table 1).

Other problems were formulated for testing the same learned schema. For instance, the second problem is the following: after assembling the side plate (`sp1`), two of the spacer pegs (`peg1`, `peg2`), the cross bar (`cb1`) and the shaft (`sft`), the robot fails to assemble the lever `lv` and realizes that the failure was due to a defect in the initial sideplate (`sp1`). The situation is similar to the one that enabled learning, but more complex, because there is a larger set of components that must be disassembled in order to replace the defective component. However, applying the planning strategy proposed above, guided by the plan skeleton contained in the learned recovery schema, an optimal solution for the new problem is easily

[3] It can be estimated as $S = B \times (B^l\text{-}1) / (B\text{-}1)$.

derived. As can be seen, at the surface, the plan for the new problem is quite different from the plan used in the episode that enabled learning. Note that, in this search, branching factors (second line in Table 1) are comparable to those of the previous case.

As Table 1 summarizes, several problems, with solution sizes varying between 15 and 40 operations, can be solved using the same basic solution (the schema in fig. 1). Variations in branching factors are minor in these problems. The key achievement of the approach, here illustrated, is that it keeps the Effective Branching Factor very near to 1. We also see that the Effective Branching Factor decreases (approaching 1.0) as the length of the target solution increases. In fact, it was nearly 1.3 when finding a plan with 15 operations, but decreased to 1.1 when finding a plan with 40 operations. This is related to the fact that, as argued before, the computational cost of the planning algorithm grows linearly with the size of the solution. The linearity claim is supported in Table 1 (3rd column), where it can be seen that the ratio between the number of nodes in the search tree and the length of the solution oscillates around 14.

Table 1 Failure recovery planning performance measures

Plan length	# nodes	# nodes / plan length	# non terminal nodes	Avg branching factor	Effective branching factor
24	374	15.6	87	4.299	1.197
32	494	15.4	114	4.333	1.143
15	160	10.7	36	4.444	1.296
21	281	13.4	64	4.391	1.217
40	564	14.1	129	4.372	1.109

5. CONCLUSIONS

In this paper, a method was proposed for solving failure recovery problems based on analogies with previous similar problems, whose basic solution was learned. An expressive semantic representation is used to guide failure recovery planning in new similar situations. The inefficiency of the classical planning-from-scratch approaches led robotics researchers to avoid artificial intelligence methods. However, as the method presented in this paper illustrates, the introduction of learning reduces planning complexity from exponential to linear in the number of required operations. This way, the task planning and failure recovery module is able to make decisions in the time scale of the normal execution of the task. This means that, if some unexpected problem arises and there is previous experience about problems of the same kind, the problem can be solved with the guarantee that the total time to complete the task won't increase significantly. Globally speaking, the approach enables the robot system to become increasingly effective, not only within a task, but also across different tasks.

REFERENCES

Camarinha-Matos, L.M., L. Seabra Lopes, J. Barata (1996) Integration and Learning in Supervision of Flexible Assembly Systems, *IEEE Trans Robotics and Automation*, vol. 12, p. 202-219.

Evans, E.Z. and C.S.G. Lee (1994) Automatic Generation of Error Recovery Knowledge through Learned Reactivity, *Proc. 1994 IEEE International Conference on Robotics and Automation*, San Diego, California, v. 4, p. 2915-2920.

Ferch, M. and J. Zhang (2001) Learning Cooperative Grasping with the Graph Representation of a State-Action Space, *Proc. 9th European Workshop on Learning Robots*, Prague, p. 65-74.

Gerevini, A. and D. Long (2005) BNF Description of PDDL3.0, October 2005.

Kedar-Cabelli, S.T. and L.T. McCarty (1987) Explanation-Based Generalization as Resolution Theorem Proving, *Proc. 4th Int'l Conf. on Machine Learning*, Irvine, CA, p. 383-389.

Kolodner, J. (1993) *Case-Based Reasoning*, Morgan Kaufmann Publishers.

Loborg, P. (1994) Error Recovery in Automation: an Overview, *AAAI'94 Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, Stanford, California, p. 94-100.

Loborg, P. and A. Törne (1994) Manufacturing Control Systems Principles supporting Error Recovery, *AAAI Spring Symposium on Detecting and Resolving Errors in Manufacturing Systems*, Stanford, CA, p. 101-108.

Lopez-Mellado, E. and R. Alami (1990) A Failure Recovery Scheme for Assembly Workcells, *Proc. 1990 IEEE International Conference on Robotics and Automation*, Cincinnati, p. 702-707.

McDermott, D., chair (1998) *PDDL: The Planning Domain Definition Language*, Yale Center for Computational Vision and Control, Tech. Rep. TR-98-003.

Meijer, G.R. (1991) *Autonomous Shopfloor Systems. A Study into Exception Handling for Robot Control*, PhD thesis, Universitaet van Amsterdam.

Minton, S. (1990) Quantitative Results Concerning the Utility of Explanation-Based Learning, *Artificial Intelligence*, vol. 42, p. 363-392.

Pettersson, O. (2005) Execution Monitoring in Robotics: a Survey, *Robotics and Autonomous Systems*, vol. 53(2), p. 73-88.

Rathmill, K. and K. Collins (1984) Development of an European Benchmark for the Comparison of Assembly Robot Programming Systems, *Proc. First Robotics Conference*, Brussels.

Seabra Lopes, L. (1999) Failure Recovery Planning in Assembly Based on Acquired Experience: Learning by Analogy, *Proc. 1999 IEEE International Symposium on Assembly and Task Planning*, Porto, pp. 294-300.

Seabra Lopes, L. and J.H. Connell (2001) "Semisentient Robots: Routes to Integrated Intelligence", *IEEE Intelligent Systems*, vol. 16, n. 5, p. 10-14.

Toguyéni, A.K.A., P. Berruet and E. Craye (2003) «Models and Algorithms for Failure Diagnosis and Recovery in FMSs», *The International Journal of Flexible Manufacturing Systems*, 15 (1), p. 57-85.

Zielinski, C. (2002) «Reaction to errors in robot systems», *RoMoCo '02. Proc Third International Workshop on Robot Motion and Control*, Bukowy Dworek, Poland, p. 201-208.