Journal of Intelligent Manufacturing, 15, 679–692, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Off-line error prediction, diagnosis and recovery using virtual assembly systems

CEM BAYDAR* and KAZUHIRO SAITOU†

Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109-2125, USA
E-mail: kazu@umich.edu

Received May 2003 and accepted February 2004

Automated assembly systems often stop their operation due to the unexpected failures occurred during their assembly process. Since these large-scale systems are composed of many parameters, it is difficult to anticipate all possible types of errors with their likelihood of occurrence. Several systems were developed in the literature, focussing on on-line diagnosing and recovery of the assembly process in an intelligent manner based on the predicted error scenarios. However, these systems do not cover all of the possible errors and they are deficient in dealing with the unexpected error situations. The proposed approach uses Monte Carlo simulation of the assembly process with the 3-D model of the assembly line to predict the possible errors in an off-line manner. After that, these predicted errors are diagnosed and recovered using Bayesian reasoning and genetic algorithms. Several case studies are performed on single-station and multi-station assembly systems and the results are discussed. It is expected that with this new approach, errors can be diagnosed and recovered accurately and costly downtimes of robotic assembly systems will be reduced.

Keywords: Off-line programming, genetic algorithms, robotic assembly systems, virtual factories, error diagnosis and recovery

1. Introduction

Automation is one of the most unavoidable concepts nowadays. The developments in the robotics area have enabled using robots in large-scale assembly operations for high productivity. However, robotic assembly systems are very sensitive to perturbations during their operation and this makes them open to unexpected failures. As stated in a previous research article (Luxhoj *et al.*, 1997), these unexpected failures cost excessive maintenance as much as 200 billion dollars in the USA in 1990.

The unexpectedness of errors arises from the fact that most of the errors are unforeseen to human experts before the operation of the line. This is natural since these systems are composed of many working parameters such as dimensional variations of the products, fixtures, sensor capabilities and robot repeatability and when these working parameters are coupled with the 3-D workspace, it is difficult to anticipate all of the error conditions with their likelihood of occurrence and 3-D-state in the workspace (Baydar and Saitou, 2001a). One example for these serious error conditions is the propagated errors, which result from the propagation of undetected errors during the assembly process. According to Abu-Hamdan and El-Gizawy (1997), error-propagation is defined as carrying an undetected error from a previous task and coupling it with another error during a proceeding task. It is also stated in Cao and Sanderson (1992) that, when a failure is detected during an operation, the operation that failed may not be the source of failure. The source of failure may have been propagated from earlier operations. Diagnosis and recovery of this type of failures is a very complex and a costly task.

*Currently with Accenture Technology Labs.
†Author for correspondence.

Previous research on the error diagnosis and recovery has focussed on either ''on-line'' investigation of error followed by a manual recovery when an error is detected or providing automated intelligent means (i.e., expert systems) to diagnose and patch the process. However, these systems are deficient to deal with most of the errors because of the following reasons:

- Not all of the error scenarios can be predicted.
- 3-D-states of the possible errors are not included (Baydar and Saitou, 2001a).
- Most of the systems are deficient in dealing with multiple error conditions (Sampath *et al*., 1996).
- Mapping the sensory domain to failure domain is not easy (Lopes and Camarinho-Matos, 1996).
- It is not easy to use heuristics for all conditions, so they are not robust (Lopes and Camarinho-matos, 1996).

Therefore, the challenge is to predict all possible error conditions as well as their likelihood of occurrence and the associated 3-D-state to provide efficient and robust error recovery means.

The proposed approach looks at the problem from a different viewpoint, which has not been used so far. It is called off-line error prediction and recovery (Baydar and Saitou, 2001b). The method uses a commercial software package to model the assembly environment virtually. After that, the possible error situations and their likelihood of occurrence are predicted by using Monte Carlo simulation of the assembly process. Having the sensory symptoms and their associated failure type and 3-D-state, these conditions are stored and used for the diagnosis using Bayesian Reasoning. Next step is using an off-line error recovery system to generate robust recovery plans (Baydar and Saitou, 2001c) that can deal with multiple error conditions of similar nature using Genetic Programming (Baydar and Saitou, 2001b; Koza, 1992). Finally, this offline recovery system can be downloaded to the controller of the robotic system to patch the process.

It is expected that the usage of this approach will decrease the lengthy ramp-up time for the testing process of the assembly systems and will provide efficient means of error recovery. It is believed that the outcomes of this approach will have impact on the industry to reduce costly downtime and maintenance expenses.

2. Previous work

Past research on error recovery in automated assembly lines has focussed on using failure trees, expert systems or other intelligent reasoning methods. Among these people, Srinivas (1997) is one of the earliest researchers who investigated error detection and recovery strategies. His approach was considering the tasks as decomposable into a sequence of transformations from the initial state to a goal state. Then the next step is building a failure tree and generating an error recovery plan.

Expert systems are also one of the most popular tools used in the error diagnosis and recovery in flexible assembly systems. Several systems were developed in the literature (Abu-Hamdan and El-Gizawy, 1997) to provide diagnosis and recovery. These are knowledge-based systems, which try to provide recovery plans for possible error conditions for multi-station assembly systems. However, they keep the description of the failure at an abstract level without including the 3-D-state and since they do not include all possible scenarios, they are deficient in handling the unexpected error situations.

Manipulating PLC codes is another approach. Zhou and DiCesare (1989), proposed four argumentation methods of process control logic code with error recovery codes: input conditioning, alternate path, feedback error recovery and feed-forward error recovery. Fuzzy reasoning was also used in conjunction with the Fuzzy Petri-Nets (Cao and Sanderson, 1992; Jing *et al*., 1996) or with expert systems (Kang and Wenhan, 1993; Tzafestas and Stamou, 1997) to provide probabilistic reasoning on the error diagnostics.

However, those approaches are deficient in handling geometric features of the assembly line, which is essential to make predictions of error scenarios. Since those 3-D error states are missing, robustness of the generated recovery plans is questionable since some unanticipated error states for the same error type (i.e., collision) may require a different plan for recovery.

The need for a robust plan was first discussed in Jennings *et al*. (1989) and an automated compliant motion planner based on geometric theory of error detection and recovery was developed. However, although the model is capable of modeling the configuration space with all of the properties such as kinematics, motion planning of the robots and 3-D positional change of the products, it is limited to one-

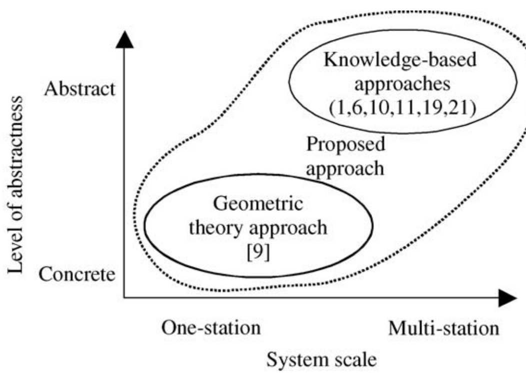


Fig. 1. Station level and system level approach.

station only. Consequently, a gap has formed between this type of concrete approach (i.e., prediction of the 3-D error states) and the abstract approach that was followed by the expert systems.

The following illustration in Fig. 1 shows the mapping of the two approaches and this gap between these two approaches. A different approach is needed to fill this gap and combine the two different types of approaches discussed above in order to provide efficient means of error recovery. The proposed method aims to fulfill this need by combining latest developments in the robotics simulation technology with the intelligent reasoning and recovery planning.

3. Proposed method

Recent developments in the computer aided robotic simulation field revealed a concept called off-line programming. In off-line programming, any robotic system can be modeled virtually in 3-D workspace and the performance of the system can be evaluated accurately from the simulations. The proposed method takes the advantage of off-line programming to predict the possible error scenarios with their 3-D-geometric states. The first step is the 3-D geometry-based modeling of an entire assembly line using a commercial modeling software package (Workspace 5 User Manual, 2000). After the system is modeled virtually, possible error scenarios are predicted using Monte Carlo simulation of assembly processes, based on the statistical model of the dimensional and functional errors in sensors, actuators, products and fixtures. Then, the next step is the off-line logic synthesis for error diagnosis and recovery from the predicted error scenarios. At this step, Bayesian reasoning is used for identifying the most probable failure while genetic algorithms (GAs) are used to generate recovery logic as discussed in our previous work in detail (Baydar and Saitou, 2001a). The reason for using genetic algorithms of genetic programming arises from the fact that most of the time same error can occur in numerous configurations in the work-space (i.e., part jamming). Genetic algorithms (or genetic programming) could help us exploit various alternatives to come up with a robust recovery code for all these configurations for the same type of error where a commonly used ‘‘manual coding’’ approach would be time consuming. The final step is building a library of recovery logic and implementing this library to the robot controller in the assembly system to patch the process against unexpected error situations. The following sections give information on the details of the each step. Figure 2 summarizes the logic of the proposed approach (Baydar and Saitou, 2001b).

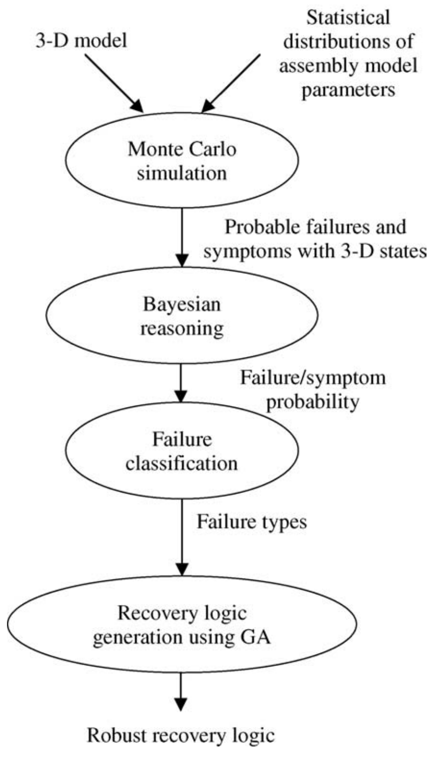

3-D model
Statistical distributions of assembly model parameters
Monte Carlo simulation
Probable failures and symptoms with 3-D states
Bayesian reasoning
Failure/symptom probability
Failure classification
Failure types
Recovery logic generation using GA
Robust recovery logic

Fig. 2. Working mechanism of the proposed approach.

3.1. *Prediction of error scenarios*

A widely used technique for simulating the errors is applying statistical methods to the tolerance analysis of mechanical assemblies. At this step, Monte Carlo simulation is used to predict the possible errors. Process parameters are sampled from the appropriate distributions and simulations are performed. The main drawback of this method is that, to get accurate estimates it is necessary to generate very large samples, which is computationally expensive. Other methods such as the quadrature and Taguchi are more systematic in their strategies for generating samples and widely used but their main drawback is that they are based on normal distribution. However, in an assembly system distributions of working parameters can be different from normal distribution. Therefore, the application of Monte Carlo method is easy and preferred because sampling from other types of distributions is applicable.

For each simulated error with relatively high likelihood of occurrence, error diagnosis logic is synthesized for effective error recovery from the error. Fundamental difference from the diagnosis in on-line cases is that a complete sequence of the events, which caused the detected error, is readily available in terms of the sampled parameters.

3.2. *Error diagnosis*

Since only providing error recovery logic is not adequate for the complete recovery process, a diagnosis system is necessary to identify the correct source(s) of error. Error diagnosis is implemented in the following way: First, from the simulation results conditional probability of each error situation is obtained for the sampled parameters. Since Monte Carlo simulation is being used, complete sequence of the events that caused an error and likelihood of each event are readily available. A reasoning engine is developed based on the symptoms (outputs from the sensory values) and the probable error conditions as it is suggested in Lopes and Camarinho-Matos (1996). This engine processes each possibility of failure and come up with most probable one (or multiple) of the five error classifications. The belief value of each type of failure is calculated by Bayesian reasoning using the following formula:

Table 1. Failure array

Failure array = {*d*, *e*, *f*, *g*, *h*}
d = grasping error
e = collision error
f = sensor failure
g = misplacement error
h = flawed parts

$$\mathrm{Bel}(F_k) = \frac{P(Y_o \backslash F_k) * P(F_k)}{\sum_{\forall l} P(Y_o \backslash F_l) * P(F_l)} \tag{1}$$

In the above formula, Y_o indicates the given symptoms from the sensor array. F_k is the type of failure from the following failure array given in Table 1 below. The formula indicates that the belief value for a failure is the ratio of this failure with the given symptoms to the sum of all other possible failures, which can occur under the same conditions. The number of elements in the failure array depends on the number of components of the assembly system.

3.3. *Error recovery*

The proposed approach provides the generation of the error recovery logic using a method called genetic programming (GP). The term genetic programming was first introduced by Koza (1992) and it uses the working principles of GAs. In GP, each member in the population is a computer program for the solution of the problem. Using an error situation obtained with the sampled parameters, a fitness function based on the allowed recovery criteria can be defined. After the definition of this fitness function, GP can be used to explore an efficient recovery algorithm.

Error recovery controller codes are robotic programs; therefore by using GP, these codes can be generated automatically. The performance of the error recovery logic is tested in a generate and test fashion (Baydar and Saitou, 2001a) such that, several recovery logic algorithms are generated with the GP engine and tested with the commercial software package. Then, the results of this evaluation are supplied to the GP engine and improved recovery logic is generated based on the obtained results (Baydar and Saitou, 2001b, 2001c). The overall algorithm is shown in Fig. 3.

In our previous work (Baydar and Saitou, 2001a), a GP module was developed because the previous

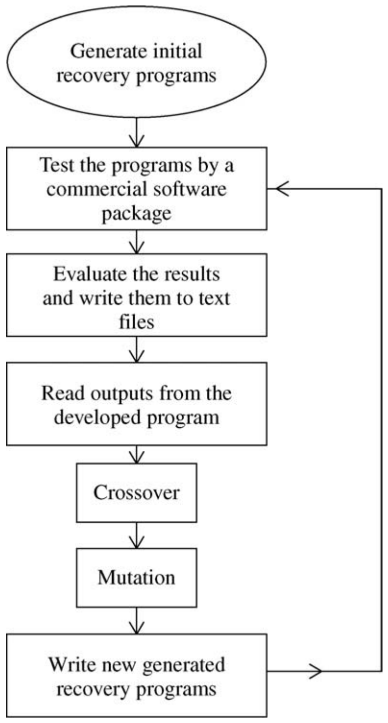

Generate initial recovery programs
Test the programs by a commercial software package
Evaluate the results and write them to text files
Read outputs from the developed program
Crossover
Mutation
Write new generated recovery programs

Fig. 3. Algorithm to generate recovery programs.

version of the software package (Workspace 4 User-Manual, 1998) was not capable of translating collision recovery path to controller codes. However, in this study, rather than GP, GAs are used to create a path for collision error recovery by representing the sequence of reference points given to the robot controller. The reason is that the recent version of the software package (Workspace 5 User Manual, 2000) has an internal engine to generate controller codes from given teach points. Therefore, a path for collision recovery is obtained first by using Genetic Algorithms. After all the points in the recovery path are obtained, Workspace is used to generate the recovery code using its internal code generation engine.

A GA engine is developed to find a common path for all collision states. The following chromosome structure is defined as shown in Fig. 4. This structure represents a position in 3-D space with three orientations. First three genes are the x, y, z coordinates while θ_1, θ_2 and θ_3 are the orientations of the workpiece in 3-D space.

The problem is formulated as a multi-level optimization problem. In the first level, the aim is

x	y	z	θ_1	θ_2	θ_3

Fig. 4. Chromosome structure.

finding a common point in 3D space, which enables the workpiece to be transported from all collision states. After that, second level is initialized by taking this common point as the initial state and finding a path to the desired position. Figure 5 shows the outline of the approach. By applying this type of a multi-level approach, the number of function evaluations is decreased since in a single-level approach all of the initial collision states must be evaluated by all members in the population. However, in this case after the intermediate state has been determined, only one state (the common point) has to be evaluated.

The objective function is defined as minimizing the Euclidian distance between the desired position (in case studies; nominal position of the peg in the hole) and the common position obtained by using the GA. Then for each state i, by taking the inverse of the objective function, the fitness function is defined as:

$$f_i = \left(\sqrt{(x_i - x_o) + (y_i - y_o) + (z_i - z_o) + (\theta_{i1} - \theta_{1o}) + (\theta_{i2} - \theta_{2o}) + (\theta_{i3} - \theta_{3o})}\right)^{-1} \tag{2}$$

The overall fitness function is defined as the summation of all individual fitness functions, evaluated for each collision error state:

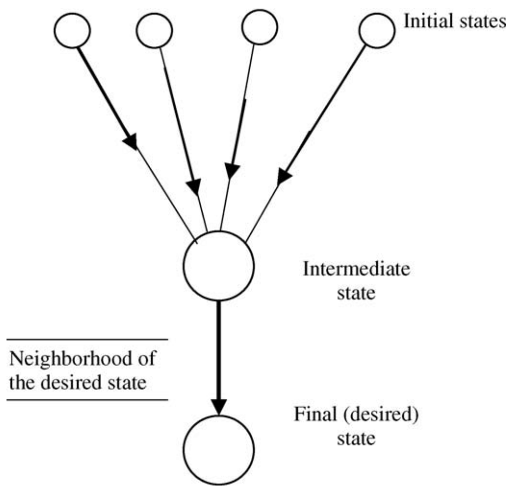


Fig. 5. Multi-level optimization approach.

Table 2. Error recovery strategies

Error type	*Strategy*
Grasping error	Try to grasp or release again
Collision error	Robust collision recovery
Sensor failure	Call maintenance
Misplacement error	Pickup the workpiece and replace
Flawed parts	Dispose the part

$$f = \sum_i f_i \tag{3}$$

In the second level since there is only one state (i.e., $i = 1$), Equation 2 is only used to calculate the fitness value of each member in the population. After the points in the error recovery path are obtained, Workspace generates a recovery code in the specified robot language. The following error recovery strategies are used for all types of errors as shown in Table 2.

4. Implementation of the approach

A virtual factory, which is composed of four modules, was developed. The main module is the virtual assembly software package, which is responsible for simulating the complete assembly process. The second module is the virtual detection module, which is used for detecting the component failures (gripper failures, sensor failures, etc.). Third module is the virtual diagnosis module for diagnosing errors. Fourth module is the virtual recovery module for applying the generated recovery codes. The factory structure is given in Fig. 6 and the detailed explanation of each module is as follows:

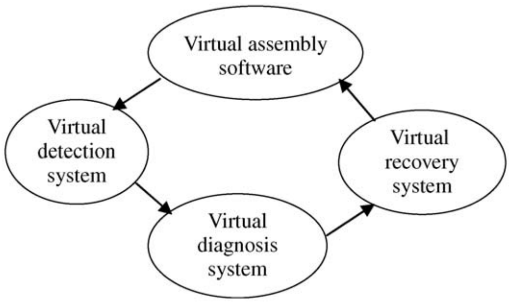


Fig. 6. Layout of the developed virtual factory.

4.1. *Structure of virtual factory*

4.1.1. *Virtual assembly software module*

This module is the commercial robotic simulation package and it includes the 3-D model of the assembly system and assembly process codes. It also contains the realistic models of assembly robots, fixtures and products to simulate the process accurately. It is also possible to detect collision errors during the assembly process since this is an implemented feature of the package.

4.1.2. *Virtual detection module*

Virtual detection module is used for detecting the errors occurred during the assembly process. In assembly systems, there are two different monitoring types. The first type is called continuous monitoring, which a parameter is monitored continuously throughout the complete assembly process. As an example, torque/force sensors are checked continuously for collision detection. The second type is called discrete monitoring, which a parameter is monitored at certain steps of the assembly process. For example grasping sensors are checked during the part picking or releasing steps to ensure the process is completed successfully. Both types of monitoring were implemented in this module. A sensor array is defined and it contains information about each sensor's state. When an error is detected, current condition of the sensor array is passed to the diagnosis module to analyze the detected error.

4.1.3. *Virtual diagnosis module*

This module uses the state of the sensor array to infer the most possible reason for the failure. However, in order to prevent incorrect automated recovery a threshold level for belief value is defined. If the diagnosed situation's belief value is greater than this threshold, system proceeds with automated recovery. If it is less than the threshold, system asks for user maintenance and the most possible failures are written into a log file.

When the system asks for user maintenance, an interactive reinforced diagnosis system is initialized. It enables the user to input further data based on the manual diagnosis by entering the identified working and non-working components during the manual inspection process. Based on this additional data, the situation can be re-diagnosed.

4.1.4. *Virtual recovery module*

This module is used for applying the recovery logic for the diagnosed failure. The outputs of the virtual diagnosis module are passed to this module and based on the failure type a strategy is followed for the recovery as discussed in Table 2.

Each strategy contains one or more recovery codes. The use of the appropriate code depends on the point where the error has been detected. Some failure types are dominant when multiple errors occurred (i.e., when a grasping failure occurred due to a flawed part and grasping, the system uses the recovery code to dispose the flawed part). Two rules were implemented to the system to recover from this type of possible scenarios. These rules are from common industry practice during the maintenance phase of combined sensor failures and flawed parts (i.e., if there is a sensor failure detected, maintenance should be patched to replace the sensor(s)).

Rule 1: IF {sensor failure} AND {any other failure} THEN ⟨call maintenance⟩

Rule 2: IF {flawed parts} AND {any other failure but NOT sensor failure} THEN ⟨replace part⟩

5. Experimental results

5.1. *Single-station assembly system*

A single-station assembly system was modeled and the performance of the approach on predicting the process capability, error diagnosis and recovery logic generation was tested. The system is composed of an IRB type robot with an inspection camera and sensors. The assembly task is inserting a peg into a hole. The sampled parameters are the statistical variations in the dimensions of peg and hole, robot repeatability (both translational ability and wrist repeatability effect), grasping ability and sensor reliability for the grasping sensor in the gripper and the position sensor located above the peg. The system was modeled using the software package as shown in Fig. 7.

The assembly process is as follows: First, a peg is grasped from the table. During this process, a camera is examining the position of peg, detecting whether it was grasped correctly or not. A sensor in the gripper is also detecting whether the peg is in the gripper or not. After the peg has been grasped, it is inserted into the hole.

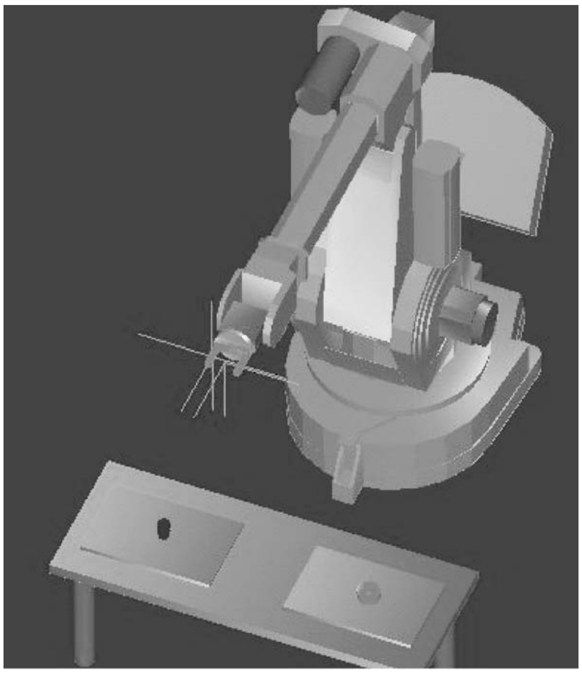


Fig. 7. Error prediction and diagnosis of a peg-in-hole problem.

During this insertion process, torque/force sensors monitor whether a collision occurred or not. In addition, during the releasing step of the peg, gripper sensor detects whether the peg is released correctly or not. The sketch of the system is given in Fig. 8.

5.1.1. *Prediction of the failures*

The first step is analyzing the process capability by predicting the possible failures. In this step, as it is suggested in a previous work (ElMaraghy *et al.*, 1988), normal distribution was assumed for the diameters of peg and hole and the parameter of robot repeatability. The parameters and tolerances are shown in Table 3. Part tolerances are taken as Class

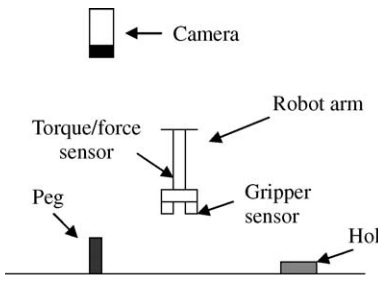


Fig. 8. Sketch of the components.

Table 3. Tolerances of the sampled parameters

Sampled parameter	*Value*
Hole diameter	50.3 ± 0.0508 mm
Peg diameter	49.92 ± 0.03175 mm
Robot repeatability	0.2 mm
Wrist angle Repeatability	0.006 deg

LC_4 (ElMaraghy *et al.*, 1988), having tolerance of the peg as h_9 and tolerance of the hole as H_{10}.

For calculating the mean and standard deviation of the normal distribution, the tolerances were taken in 3σ-range. The distribution values of the sampled parameters are given in Table 4 below:

The number of simulations was taken as 1000 and the process capability was investigated. Out of 1000 simulations 317 errors occurred. Table 5 shows the prediction of the possible failures:

It was found out that the process capability is 69% based on the 317 errors occurred out of 1000 simulations.

5.1.2. *Error diagnosis system*

An error diagnosis system was built for the assembly system. It uses the symptoms obtained from the assembly system and calculates the likelihood of each possible failure based on these symptoms, coming up with most probable failure type(s). The sensor and failure arrays are given in Table 6.

The sensor values can get values of 0 or 1 depending on their activeness. For example if $(0, 1, 0)$ is received from the sensor array, this means that torque/force sensor has detected a collision. Similarly, if $(0, 0, 1)$ is received, this means that the inspection camera has detected a missing part in the assembly. The values for the sensor and failure arrays are given in Table 7 below:

Using the results of Monte-Carlo simulation obtained in the previous step, a belief value was calculated for each failure type. For each probable symptom (sensory input), the most probable failure type and its probability (belief value) are given in Table 8.

Table 4. Distribution values of the sampled parameters

Sampled parameter	*Distribution type*
Peg	Normal $(0, 0.0106\,\text{mm})$
Hole	Normal $(0, 0169\,\text{mm})$
Robot repeatability	Normal $(0, 067\,\text{mm})$
Wrist angle repeatability	Normal $(0, 0.002\,\text{deg})$
Grasping ability	Uniform (0.9 probability)
Position sensor	Uniform (0.95 probability)
Gripper sensor	Uniform (0.95 probability)

Table 5. Failure types and percentages

Failure type	*Percentage*
Collision error	36.6
Grasping error (picking)	27.44
Grasping error (releasing)	27.1
Misplacement error	0
Sensor failure	0
Flawed parts	0
Collision + sensor failure	5.397
Gripper + sensor failure	3.45

Table 6. Sensor and failure arrays

Sensor array $\{a, b, c\}$	*Failure array* $\{d, e, f, g, h\}$
a = gripper sensor	d = grasping error
b = torque/force sensor	e = collision error
c = camera	f = sensor failure
	g = misplacement error
	H = flawed parts

Table 7. Sensor array codes failure types and associated values

	Values
Sensor type	
Gripper sensor	0-none, 1-no part in the gripper
Torque/force sensor	0-none, 1-collision detected
Camera	0-none, 1-incomplete assembly
Failure type	
Grasping error	0-none, 1-picking, 2-releasing
Collision error	0-none, 1-collision
Sensor failure	0-none 1-gripper sens., 2-camera, 3-both
Misplacement error	0-none, 1-misplacement
Flawed parts	0-none, 1-flawed part

Table 8. Results of the diagnosis process

Symptom	*Failure type/probability*
$(1, 0, 0)$	$(2, 0, 0, 0, 0)/0.996$
$(0, 1, 0)$	$(0, 1, 0, 0, 0)/0.989$
$(0, 0, 1)$	$(1, 0, 0, 1, 0)/1$
$(1, 0, 1)$	$(1, 0, 0, 0, 0)/1$

5.1.3. *Recovery code generation system*

A recovery code generation system was developed and generation of recovery logic for collision errors occurred between peg and hole was experimented. During the simulation of assembly process, 133 collision states have been identified between peg and hole. In this step, the aim is to generate one robust recovery code which can be used for all of these collision states. In the first level of optimization, these states were taken as starting points. The aim is to find a common point (an intermediate state) in the 3-D space, which the robot arm can reach from any collision point. Population size was taken as 100, while crossover and mutation probabilities were defined as 0.9 and 0.05 respectively. After nine generations, a common point (intermediate state) was found. The configuration of this point in 3-D-space was obtained as (1553.09, 533, 770.321, 0, 78, 0). The history of optimization is given in Fig. 9.

Now taking the intermediate state as the starting point, second level optimization was initiated (Fig. 10). This time the objective is finding a path to the reach the desired position from the point obtained in the first step. After five generations, the system was able to find a point which is close to the desired position. This final state was identified as (1553.79, 530.89, 722.5, 0, 90, 0).

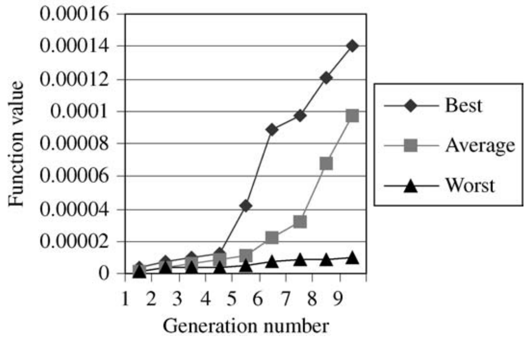


Fig. 9. Optimization history of the first level.

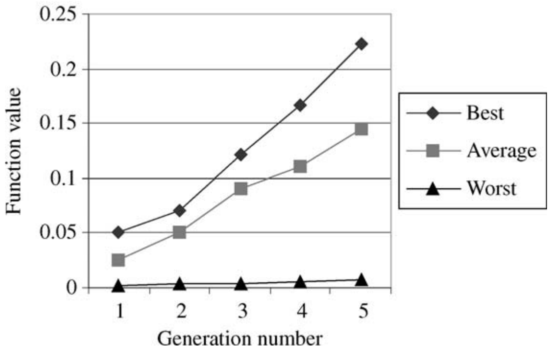


Fig. 10. Optimization history of the second level.

Therefore, the overall path is composed of two points and they are the intermediate state (1553.09, 533, 770.321, 0, 78, 0) and the final state (1553.79, 530.89, 722.5, 0, 90, 0) respectively. Note that the desired point in the configuration space is (1553.89, 531, 718, 0, 90, 0) and the final state is very close to the desired position.

After obtaining the path points, recovery code for collision error was generated in RAPID language using Workspace v.5's translation tools. The following shows the generated collision recovery code. The strategy for the recovery was identified as retracting the robot arm to the common point, then inserting the peg into the hole.

5.1.4. *Case study—diagnosis and recovery of a collision error*

The performance of the system was tested with a collision error scenario occurred between peg and hole. Figure 11 shows a collision instance experienced during the assembly process.

After this collision is detected, system enters into the diagnostic mode. An automatic recovery threshold value was defined as 0.85, meaning that diagnosed belief values, which are greater than this threshold, will be automatically recovered. If the diagnosed belief value is less than this threshold, the system asks for human maintenance. Based on the given symptom from the torque/force sensor, the diagnosis system identified the error as a collision error with a belief value of 0.989. After that, system proceeds with error recovery procedure by applying the collision recovery code generated in the previous section. It was validated that the generated recovery logic is

```
MODULE collision recovery
CONST robtarget NewGP005 := [[1553.09, 533, 770.321], [0.707107, 8.65927E - 017,
0.707107, 8.65927E - 017], [0, 0, 1, 0], [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];
CONST robtarget NewGP006 := [[1553.79, 530.89, 722.5], [0.707107, 8.65927E - 017,
0.707107, 8.65927E - 017], [0, 0, 1, 0], [9E9, 9E9, 9E9, 9E9, 9E9, 9E9]];
PERS tooldata t_Gripper := [ TRUE, [[0, 2.27777E - 014, 93], [1, 0, 0, 0]],
[0.01, [0.01, 0.01, 0.01], [1, 0, 0, 0], 0, 0, 0]];
PERS tooldata t_Nil := [TRUE, [[0, 0, 0], [1, 0, 0, 0]], [0.01, [0.01, 0.01, 0.01],
[1, 0, 0, 0], 0, 0, 0]];
PERS wobjdata w_Nil := [FALSE, TRUE, " ", [[0, 0, 0], [1, 0, 0, 0]], [[0, 0, 0],
[1, 0, 0, 0]]];
PROC Path1()
MoveJ NewGP005, v1000, z1, t_Gripper;
MoveJ NewGP006, v1000, z1, t_Gripper;
!- ThisDocument.RunBehavior "Gripper", "OpenGripper", " "
!- ThisDocument.RunBehavior "Gripper", "UnGrasp", " "
ENDPROC
PROC main()
Path1;
ENDPROC
ENDMODULE
```

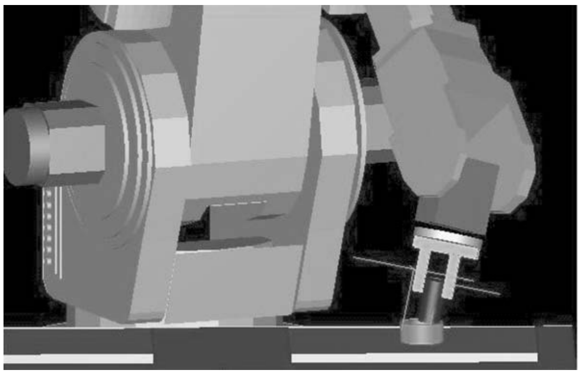


Fig. 11. Collision error case.

successful to recover the collision error. The recovered position is shown in Fig. 12.

This case study demonstrated the performance of the proposed approach for single-station systems. The sensory information is mapped on the failure domain efficiently, to predict the probable failures and their 3-D state. Another advantage is that the generation of the recovery code using the virtual assembly system saves time on finding a robust recovery logic algorithm.

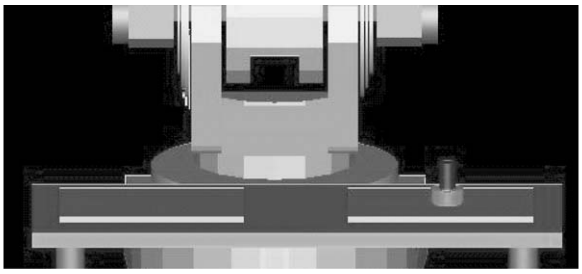


Fig. 12. Assembly system after applying the recovery algorithm.

5.2. *Multi-station assembly system*

A multi-station assembly system, which is responsible for mounting and welding workpieces together, was modeled using Workspace (Workspace 4 User-Manual, 1998) and shown in Fig. 13.

The system is composed of three IRB 6400-type

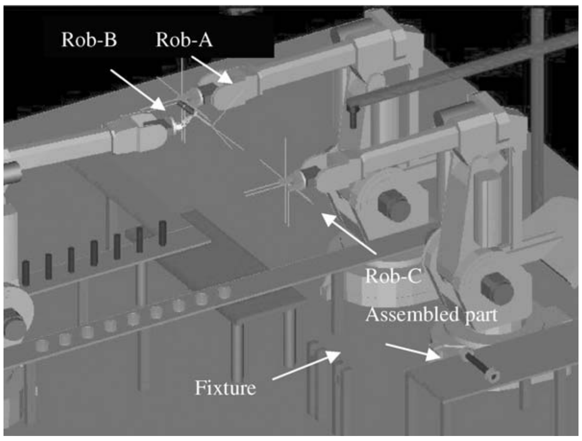


Fig. 13. Collision error case.

industrial robots. The assembly process is as follows: At first, Rob-A picks up the cylindrical piece from the conveyor and inserts it into the hole of the second piece on the second station. Then, the welding robot (Rob-B) approaches the two assembled pieces and welds those pieces together. After that, Rob-C picks up the welded piece and places it on the fixture and all parts are welded together by Rob-B. Finally the complete assembly is transferred on the conveyor by Rob-C. During the assembly operation, two inspection cameras are used for the verification process. These cameras were placed over the first station where Rob-A picks up the cylindrical piece and over the fixture respectively.

The elements in the modeled sensor array and their signal codes are given in Tables 9 and 10. Each sensor parameter, except the cameras, gets 0 or 1 depending on the signal feedback from the sensors. If there is a signal indicating an abnormal situation, then the associated parameter gets 1. For the inspection cameras the following codes were implemented as shown in Table 12.

Table 9. Parameters for sensor array

Sensor array = $\{A, B, C, D, E, F\}$
A = gripper sensor of Rob-A
B = torque/force sensor of Rob-A
C = camera over Station 1
D = gripper sensor of Rob-B
E = torque/force sensor of Rob-B
F = camera over the fixture

Table 10. Camera codes

1 = workpiece not grasped or released
2 = incomplete assembly
3 = no parts
4 = part jamming

The failure array was designed to provide information on the possible failure types as discussed in Table 1. As shown in Table 11, all of the possible failure types of grasping and sensor failures were implemented to the model. In grasping, failure code 6 is different from 7 in the sequence of failures.

Since sequence of failures is also important for the

Table 11. Failure codes

Grasping failure codes	*Sensor failure codes*	*Collision failure codes*
1: Rob-A gripper picking	1: Camera 1	1: Collision at Station 1
2: Rob-A gripper releasing	2: Gripper A	2: Collision at Station 2
3: Rob-C gripper picking	3: Camera 2	3: Collision at fixture
4: Rob-C gripper releasing	4: Gripper B	
5: (Rob-A + Rob-C) picking	5: Cam.1, Grip.A	
6: Rob-A pick + Rob-C release	6: Cam.1, Cam.2	
7: Rob-C release + Rob-A pick	7: Cam.1, Grip.B	
8: (Rob-C + Rob-A) release	8: Grip.A, Cam.2	
	9: Grip.A, Grip.B	
	10: Cam.2, Grip.B	
	11: Cam.1, Cam.2, Grip.A	
	12: Cam.1, Grip.A, Grip.B	
	13: Cam.1, Cam.2, Grip.B	
	14: Grip.A, Grip.B, Cam.2	
	15: All of the sensors	

Table 12. Sampled parameters and values

Parameter	*Nominal value/Distribution*
Robot Repeatability	0.02 mm/Normal $(0, 0.067\,\text{mm})$
Grasping ability	Uniform (0.9)
Gripper sensor	Uniform (0.99)
Inspection Camera	Uniform (0.99)
Peg	49.92 mm/Normal $(0, 0.0106)$
Hole	50.3 mm/Normal $(0, 0169)$

appropriate recovery, these two codes are different from each other.

Several parameters were sampled from the assembly process. These parameters include robot repeatability for each robot, gripper reliability, gripper sensors reliability, sensor reliability for the inspection cameras and dimensional tolerances of each piece. Each parameter and its distribution type are given in Table 12. The values in the parenthesis indicate the mean and the standard values of the associated parameter.

Complete assembly process was simulated off-line 50,000 times. The threshold level of the belief value for automated recovery was taken as 0.8. During this simulation process several types of error-propagation were observed (Baydar and Saitou, 2001c). One example is discussed below.

5.2.1. *Propagation error resulted in part jamming at Station-1*

In this case, a jamming error is detected at Station-1 by the torque/force sensor of Rob-A in the form of a sensory array input $(0, 1, 0, 0, 0, 0)$. The reason for this failure is that Rob-A did not release the piece in its gripper and Camera-1 and Rob-A's gripper sensor were both malfunctioning to detect this initial error. Furthermore, this error has been propagated to the next stage of the assembly process and coupled with the Camera-2 failure to detect the incomplete assembly. When the next cycle has started, the next workpiece collided with the previous part still held in Rob-A's gripper. The failure situation is given in Fig. 14.

Information from the sensor array was diagnosed by the Virtual Diagnosis module as shown in Fig. 15. The diagnosed failure reason and its belief value are as follows (Table 13).

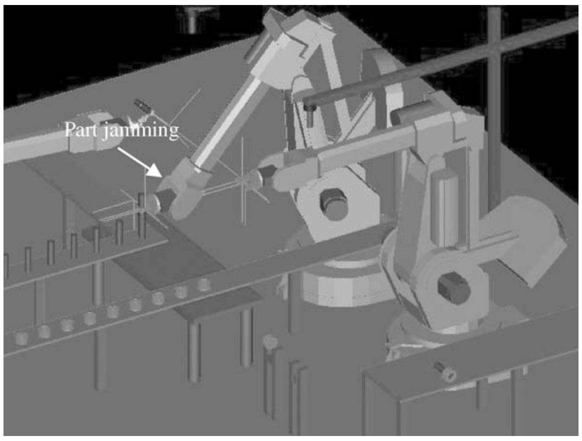
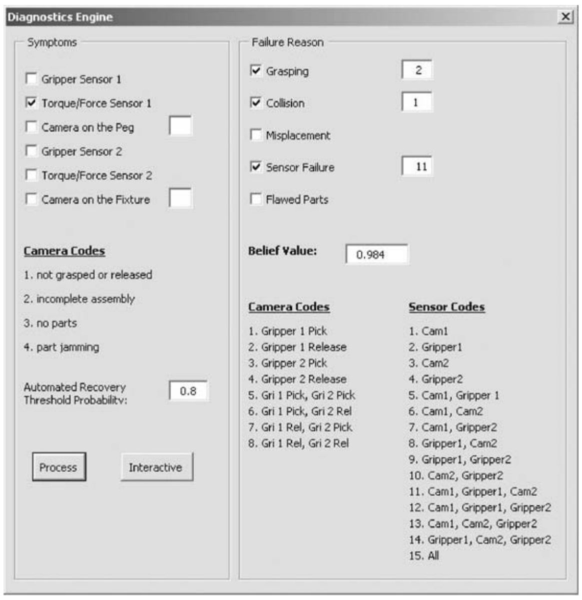


Fig. 14. Part jamming at the 1st station.



Diagnostics Engine

Symptoms

- [ ] Gripper Sensor 1
- [✓] Torque/Force Sensor 1
- [ ] Camera on the Peg
- [ ] Gripper Sensor 2
- [ ] Torque/Force Sensor 2
- [ ] Camera on the Fixture

Camera Codes

1. not grasped or released
2. incomplete assembly
3. no parts
4. part jamming

Automated Recovery Threshold Probability: 0.8

Failure Reason

- [✓] Grasping 2
- [✓] Collision 1
- [ ] Misplacement
- [✓] Sensor Failure 11
- [ ] Flawed Parts

Belief Value: 0.984

Camera Codes	**Sensor Codes**
1. Gripper 1 Pick	1. Cam1
2. Gripper 1 Release	2. Gripper1
3. Gripper 2 Pick	3. Cam2
4. Gripper 2 Release	4. Gripper2
5. Gri 1 Pick, Gri 2 Pick	5. Cam1, Gripper 1
6. Gri 1 Pick, Gri 2 Rel	6. Cam1, Cam2
7. Gri 1 Rel, Gri 2 Pick	7. Cam1, Gripper2
8. Gri 1 Rel, Gri 2 Rel	8. Gripper1, Cam2
	9. Gripper1, Gripper2
	10. Cam2, Gripper2
	11. Cam1, Gripper1, Cam2
	12. Cam1, Gripper1, Gripper2
	13. Cam1, Cam2, Gripper2
	14. Gripper1, Cam2, Gripper2
	15. All

Fig. 15. Virtual diagnosis output.

Table 13. Output of virtual diagnosis module (symptom array $(0, 1, 0, 0, 0, 0)$)

Diagnosed failure	
Belief value	0.984
Grasping failure	Rob-A gripper release failure
Sensor failures	Cam.1, Rob-A gripper, Cam.2

The proposed failure reason's belief value is greater than the threshold so the system proceeds with automated recovery. The suggested failure recovery strategy is calling the maintenance person to replace the malfunctioning sensors and components. Therefore, at this point the system dispatches a maintenance expert and supplies the information about the sensory components to be replaced.

This case study revealed the fact that although propagated errors occur in less likelihood, they cannot be avoided. The modeled system is composed of relatively less number of components when compared to the large-scale auto-body or consumer electronics assembly lines; however it is still difficult to analyze the system. Therefore, the developed Virtual Factory aids on prediction, diagnosis and recovery of the complex errors, which may propagate during the assembly process.

6. Conclusions

A new approach on the investigation of error prediction, diagnosis and recovery was discussed in this paper. The diagnosis and recovery tasks of assembly errors are complex since they cannot be predicted easily prior to the operation of the assembly line. Several methods have been used in the literature to predict the possible propagation of undetected errors using failure propagation trees, failure mode and effect analysis or using fuzzy logic. However, they are deficient in anticipating all errors and they leave the 3-D-states of the possible errors out of consideration, which makes the generated recovery codes non-robust. Because of these facts, the need was identified as to predict all possible error conditions as well as their likelihood of occurrence and the associated 3-D-states to provide efficient and robust error recovery means.

The discussed approach uses a commercial software package for robotic simulation for the prediction, diagnosis and recovery of the possible failures in four steps:

(1) Modeling of the assembly system using a commercial off-line robotic software package.
(2) Monte Carlo simulation of assembly processes to predict the possible error conditions and their likelihood of occurrences.
(3) Logic synthesis for error diagnosis and recovery from the predicted error scenarios based on the three-dimensional model using Bayesian Reasoning and Genetic Algorithms.
(4) Downloading the developed recovery codes to the robotic controller to patch the assembly process against the unexpected errors.

A case study was conducted by modeling a single-station assembly system, which is composed of a peg-in-hole assembly process. The obtained results showed that, the system is capable of identifying the possible failures and their 3-D geometrical states. Based on these failure scenarios, the system is also capable of generating robust recovery codes as discussed in our previous work (Baitou and Saitou, 2001a). A second case study was completed focussing on a multi-station assembly process for diagnosing propagated errors. The obtained results showed that the method is capable of predicting propagated errors and it is efficient to diagnose even the failure case is too complex to solve for a human expert.

One of the major disadvantages of the discussed approach is that the costly computational time of performing Monte-Carlo simulations. Also, a high threshold value should be selected for automatic recovery in order to prevent error recovery problems originated from misdiagnosis. We believe that another possible implication may be the calibration of the virtual factory with the actual system but this disadvantage can be overcome with the aid of developments in the virtual assembly software area.

Although the proposed approach has not been implemented into an actual system yet, the future work includes testing it with a real assembly system and validating the accuracy of the results. It is expected that the usage of this approach will decrease the lengthy ramp-up time for the testing process of the assembly systems and will provide efficient means of error recovery. We believe that the outcomes of this approach will have impact on the industry to reduce costly downtime and maintenance expenses.

Acknowledgments

The authors are grateful for the technical support of Flow Software Inc. for the Workspace$^{\text{TM}}$ software. We also acknowledge Rackham Graduate School of

Studies at The University of Michigan for supporting this work.

References

Abu-Hamdan, M. G. and El-Gizawy, A. S. (1997) Computer aided monitoring system for flexible assembly operation. *Computers in Industry*, **34**, 1–10.

Baydar, C. and Saitou, K. (2001a) Automated generation of error recovery logic in assembly systems using genetic programming. *Journal of Manufacturing Systems*, **20**(1), 55–68.

Baydar, C. and Saitou, K. (2001b) Off-line error prediction, diagnosis and recovery using virtual assembly systems. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*.

Baydar, C. and Saitou, K. (2001c) Prediction and diagnosis of propagated failures in assembly systems using virtual factories. *Proceedings ASME Design Engineering Technical Conferences—Computers in Engineering*.

Cao, T. C. and Sanderson, A. C. (1992) Sensor-based error recovery for robotic task sequences using fuzzy petri-nets. *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, **2**, 1063–1069.

Chang, S. J., DiCesare, F. and Goldbogen, G. (1991) Failure propagation trees for diagnosis in manufacturing systems. *IEEE Transactions on System Man Cybernetics*, **21**(4), 767–776.

ElMaraghy, H. A., ElMaraghy, W. H. and Knoll, L. (1988) Design specification of parts dimensional tolerance for robotic assembly. *Computers in Industry*, **10**, 47–59.

Evans, E. Z. and Lee, S. G. (1994) Automatic generation of error recovery knowledge through learned activity. *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, **4**, 2915–2920.

Jennings, J., Donald, B. and Campbell, D. (1989) Towards experimental verification of an automated compliant motion planner based on a geometric theory of error detection and recovery. *Proceedings of the IEEE International Conference on Robotics and Automation*, 632–637.

Jing, Q., Xisen, W., Zhihua, P. and Youngcheng, X. (1996) A research on fault diagnostic expert system based on fuzzy petri nets for FMS machining cell. *Proceedings IEEE International Conference on Industrial Technology*, 122–125.

Kang, L. and Wenhan, Q. (1993) Fuzzy expert system in robotic assembly workcell. *Proceedings IEEE TENCON*, 738–741.

Koza, J. R. (1992) *Genetic Programming: On the Programming of Computers by Natural Selection*, MIT Press, Cambridge, MA.

Lam, R. K., Pollard, N. S. and Desai, R. S. (1990) Studies in knowledge-based diagnosis of failures in robotic assembly. *Proceedings of the IEEE Conference on Robotics and Automation*, 60–65.

Lopes, L. S. and Camarinho-Matos, L. M. (1996) Towards intelligent execution supervision for flexible assembly systems. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1225–1230.

Lunze, J. and Schiller, F. (1999) An example of fault diagnosis by means of probabilistic logic reasoning. *Control Engineering Practice*, **7**, 271–278.

Luxhoj, J. T., Riis, J. O. and Thorsteinsson, U. (1997) Trends and perspectives in industrial maintenance management. *Journal of Manufacturing Systems*, **16**(6).

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K. and Teneketzis, C. (1996) Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, **4**(2), 105–124.

Srinivas, S. (1997) *Error Recovery in Robot Systems*. Ph.D. Thesis, California Institute of Technology.

Tzafestas, S. and Stamou, G. B. (1997) Concerning automated assembly: knowledge-based issues and a fuzzy system for assembly under uncertainty. *Computer Integrated Manufacturing Systems*, **10**(3), 183–192.

Visinsky, M. L., Cavallaro, J. R. and Walker, I. D. (1994) Expert system framework for fault detection and fault tolerance in robotics. *Computers in Electrical Engineering*, **20**(5), 421–435.

Workspace 5 User-Manual. (2000) Flow Software, Inc.

Workspace 4 User-Manual. (1998) Flow Software, Inc.

Zhou, M. C. and DiCesare, F. (1989) Adaptive design of petri-net controllers for error recovery in automated manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics*, **19**(5), 963–973.