Journal of Manufacturing Systems
Vol. 20/No. 1
2001

Automated Generation of Robust Error Recovery Logic in Assembly Systems Using Genetic Programming

Cem M. Baydar and **Kazuhiro Saitou,** Dept. of Mechanical Engineering and Applied Mechanics, University of Michigan, Ann Arbor, Michigan, USA

Abstract

Automated assembly lines are subject to unexpected failures, which can cause costly shutdowns. Generally, the recovery process is done "on-line" by human experts or automated error recovery logic controllers embedded in the system. However, these controller codes are programmed based on anticipated error scenarios and, due to the geometrical features of the assembly lines, there may be error cases that belong to the same anticipated type but are present in different positions, each requiring a different way to recover. Therefore, robustness must be assured in the sense of having a common recovery algorithm for similar cases during the recovery sequence.

The proposed approach is based on three-dimensional geometric modeling of the assembly line coupled with the genetic programming and multi-level optimization techniques to generate robust error recovery logic in an "off-line" manner. The approach uses genetic programming's flexibility to generate recovery plans in the robot language itself. An assembly line is modeled and from the given error cases an optimum way of error recovery is investigated using multi-level optimization in a "generate and test" fashion. The obtained results showed that with the improved convergence gained by using multi-level optimization, the infrastructure is capable of finding robust error recovery algorithms. It is expected that this approach will require less time for the generation of robust error recovery logic.

Keywords: *Automated Assembly Systems, Error Recovery, Genetic Programming, Multi-Level Optimization*

Introduction

An unexpected failure is an unavoidable phenomenon, which causes the automated assembly lines to halt their operation. These failures can bring out drastic results in economical issues. As indicated in the results of the EUREKA project,[1] initiated to benchmark maintenance in Scandinavian countries in 1992, approximately 30% of the time spent on maintenance is used for unforeseen repairs, 20% for preventive maintenance, and 37% for planned repairs. A similar survey in the United States showed that excessive maintenance costs were approximately $200 billion in 1990.

The diagnosis and recovery from such failures are normally handled by on-line investigation of the assembly line by human experts, which means costly shutdown of the assembly lines. Another approach is using controller codes. It is stated in Zhou and DiCesare[2] that in automated systems up to 90% of the control coding effort is based on error recovery by using programmable logic controller (PLC) codes. However, these PLC codes are programmed by humans based on "expected" error scenarios and are deficient in dealing with "unexpected" scenarios, leaving the recovery process to manual labor work. A novel approach[3] to deal with the unexpected failures is off-line synthesis of error diagnosis and recovery logic based on the three-dimensional geometry-based modeling of an entire assembly line. Generation of unexpected error cases can be accomplished by using Monte Carlo simulation of the assembly process based on the statistical model of the dimensional and functional errors in sensors, robots, and products. Once those unexpected cases are generated, error recovery logic synthesis for those cases can be studied.

It is stated in Visinsky et al.[4] that in an assembly line most errors occur during the part transport and part mating. However, because the part presentation errors are mostly dependent on the nature of the assembly line,[5] there may be different ways of recovery. Due to the geometrical nature of the assembly lines, there can be errors, which have the same error type (that is, collision) but need to be recovered by using a procedure different from the anticipated case. For example, a collision error may occur in many different ways during an assembly process. The diagnosis of this failure can reveal the cause of the error correctly. However, it may not be possible

to detect the exact location of the collision, which is very important for the recovery procedure. If the location is different from the anticipated place, the implemented recovery logic algorithm may not be useful. Therefore, robust ways of recovery logic must be investigated.

The solution of this problem consists of a special program, which is composed of several commands and directions for the industrial robot. This program can be downloaded to the robot controller to perform the recovery task. These commands in the recovery program are chosen from an available set of commands that makes the problem a discrete decision-making process.

In this paper, a new approach on generating robust recovery logic is presented. The proposed approach uses multi-level optimization to generate *robust* recovery algorithms for the given error cases. The studied error cases are all taken from collision errors occurred during a part placement process. The generation of the optimal recovery program is done by a heuristic search[6] among the alternative error recovery programs and is called genetic programming. Genetic programming (GP)[7] aspires to induce a population of computer programs that improve automatically as they experience the data on which they are trained.

The difference of this method from the previously developed robot path planning systems[8-10] is that previous methods require an initial error state to develop a recovery plan and these initial states are tried to be anticipated. Once a plan is developed with a planner system, it requires translation of the generated plan to working controller codes. However, one of the advantages of the proposed approach is that it uses genetic programming to generate efficient recovery plans in the robot language itself. Therefore, no post-processing is required to convert the plans into controller codes. Besides, the approach is coupled with a robotic simulation software package, which enables simulation of unexpected errors.

The method of evaluating the optimal program is done in a "generate and test" fashion, and the validity of the generated programs is tested with a commercial software package[11] called Workspace from Flow Software Inc. The following section contains information on the previous work done on error recovery and diagnosis, highlighting the importance of having robust recovery logic in recovery systems.

Previous Work

Srinivas[12] is one of the earliest researchers who investigated error detection and recovery strategies. His approach was considering the tasks to be decomposable into a sequence of transformations from the initial state to a goal state. Each of the states between the initial state and the goal state is monitored. If a state has not been reached, this means a failure has occurred. The next step is building a failure tree and generating an error recovery plan.

Expert systems are one of the most popular tools used in error detection and recovery in flexible assembly systems. Two different methods are used in the literature. The first method[13] uses an expert system to monitor robot operations, and if it detects an error, the robot stops. The expert system goes into an error diagnostic mode and analyzes the sensors that describe the environment. After finding the cause of the error, the expert system recommends a solution procedure. The second method[14] uses the expert system to plan and execute the assembly process. The user enters a part-oriented description of the assembly. The expert system generates an assembly plan, and if an error occurs, it revises the plan. Tzafesas and Stamaou's approach[15] is based on using knowledge-based approaches for automated assembly. It is believed that fuzzy logic and fuzzy reasoning based techniques are more adequate than the others under specific circumstances. The assembly system is trained on-line for the possible errors, and appropriate recovery logic rules are embedded to the system. Recovery is accomplished by making deductions out of these rules. Telagi and Soni[16] investigated the different control methodologies such as neural networks and fuzzy logic in a manufacturing environment. Evans and Lee[17] developed a method to integrate reactive planning for automated error recovery, while Kao[18] discussed the selection of an optimal error recovery strategy among the given recovery options using a semi-Markovian model of production states during recovery operations.

Several works have also been done on how to manipulate the PLC code safely with the error recovery codes generated manually or automatically without introducing new errors. Zhou and DiCesare[2] proposed four argumentation methods of process control logic code with error recovery codes: input conditioning, alternate path, feedback error recovery, and feedforward error recovery. Cao and Sanderson[19] proposed a fuzzy Petri net controller as

an integrated representation of process control logic and error recovery. However, those approaches are lack of handling geometric features of the assembly line, which is essential to make predictions of error scenarios.

The need for robust recovery logic arises after the error diagnosis stage. Error diagnosis is the key step before determining the recovery process. Complete diagnosis must be performed for the efficient error recovery. The established techniques of failure mode and effect analysis (FMEA), fault tree analysis (FTA), and event tree analysis (ETA) have been in use for many years.[20] FMEA is used to examine all possible component failures and to identify their first order and final effects on the system. FTA and ETA may be applied at various levels for examining the errors and failures in a system. FTA is a top-down technique for assessing the way in which several failures can cause a single outcome or a system failure. ETA is a forward technique, which may be used to examine the propagation of an initiating event (or failure) with the presence of a number of other events, failures, faults, or conditions.

Abu-Hamdan and El-Gizawy[14] developed a knowledge-based system for monitoring, diagnosis, and error recovery for flexible assembly operations. The control system consists of a distributed network of intelligent sensing, action, and reasoning agents. For error diagnosis, an AND/OR type failure tree is constructed. The error type is the goal node (root of the tree at the top level). The error causes are the subgoals of the tree. The facts of the errors (that is, sensor failure) are represented as the leaves of the subgoals. The use of fault trees as a database of run-time fault detection is discussed in Visinsky et al.[4] An expert system is embedded to the system to monitor the faults and maintain the probability of failure for each node within the tree. Two finite state machines (FSM) are used. The user/executive FSM handles the interaction between the user and the robot, while the critic FSM is responsible for the safety of the robot system. Two other proposed methods are known as failure reason analysis (FRA) and multiple outcome analysis (MOA), which are discussed in Hardy, Barnes, and Lee.[21] FRA is based on finding an explanation of the failure and tries to derive a plan for recovery by using a failure tree. The tree contains action nodes and failure nodes. The data about the type of the error are collected from the tree and passed to a planner module. In MOA, the states of the workcell are in consideration. Detecting the deviation of the states from the expected ones reveals the fact of failure. After an error is detected, available data and gathered data are used to conclude a predefined recovery strategy.

Several systems were developed for recovery planning in manufacturing and assembly systems. Kis and Vãncza's system[22] involves combination of expert systems with genetic programming using STRIPS operators in manufacturing domain. Another recent planning tool is developed by Klein, Jonsson, and Backstrom[23] for the error recovery process in assembly systems. The system is composed of a planner using a polynomial time-planning algorithm and a translator that translates the plans into GRAFCET charts. However, this system requires post-processing to compile the recovery algorithm into a code for a programmable logic controller.

All of the systems discussed above are focused on the on-line recovery and training of the assembly lines, which makes the process time and money consuming. Another disadvantage is performing diagnosis and recovery using expected error cases because, due to the geometrical nature of the assembly lines, there can be errors that have the same error type (such as collision) but need to be recovered by using a procedure different from the anticipated case. For example, a collision error may occur in many different ways during an assembly process. The diagnosis of this failure with the developed systems discussed above can reveal the cause of the error correctly. However, it may not be possible to detect the exact location of the collision, which is very important for the recovery procedure. If the location is different from the anticipated place, the implemented recovery logic algorithm may not be useful. Therefore, robust ways of recovery logic must be investigated.

Proposed Method

For robots to execute the programs generated off-line successfully, the dimensions of the real-world components need to be modeled accurately. Otherwise, the difference will lead to positioning and orientation errors. The method presented here is based on an accurate three-dimensional geometric model of the assembly line by using a commercial software package. This three-dimensional model

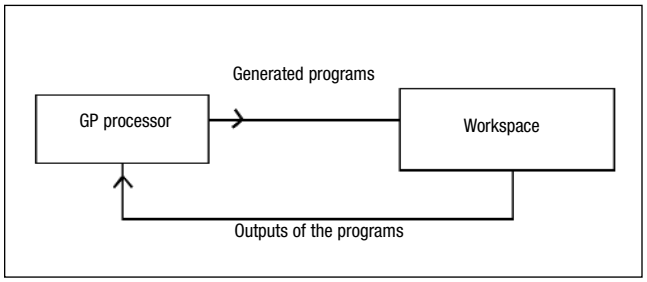

Figure 1
Framework of the Optimizer

provides the framework of testing the generated error recovery logic off-line in less time than the conventional on-line methods.

The system[3] is composed of a software module and a commercial robotic simulation package. The software module is responsible from the evolutionary computation process. This module enables the user to generate programs for the problem and after that evaluate the outputs from the commercial software and proceed with the evolution process for the next generations. The basic working mechanism is given in *Figure 1*.

Problem Definition

In this paper, recovery from collision errors is studied. Therefore, the objective is defined as to minimize the part placement error between the final position and its desired position on the fixture. A distance function between the recovered position and the desired position is used for the objective function. Variables x_o, y_o, and z_o are the desired coordinates in 3-D space, while the coordinates obtained by a recovery program are presented as x, y, and z. Other constraints such as collisions or the working envelope of robot are handled internally by the software package and cannot be formulated explicitly. To be conservative, part tolerances are taken as minimum clearance during the recovery logic generation process. Therefore, the problem is a single objective optimization problem with the objective function:

$$\text{Minimize } \sqrt{(x-x_o)^2+(y-y_o)^2+(z-z_o)^2} \qquad (1)$$

During the recovery procedure, tolerances are allowed for the final position of the workpiece. The acceptable tolerances are defined as 5 mm in all dimensions. These tolerances are obtained from the acceptable assembly tolerance for the workpieces. This also means that any local optimum, which is below 5 mm difference in all coordinates, is acceptable as the solution of the problem.

$$|x-x_o| \le 5 \qquad (2)$$

$$|y-y_o| \le 5 \qquad (3)$$

$$|z-z_o| \le 5 \qquad (4)$$

The problem is handled in two phases:[24]

- Solving a relaxed problem for n different error states to reach an intermediate state.
- From the obtained intermediate state, solving the original problem to reach a desired state.

In the first step, several collision cases are solved in parallel to find a recovery algorithm, which enables reaching a common state for all of the cases. For this step, the same objective function is used, but the constraints are relaxed from 5 mm to 15 mm. A feasible working envelope is defined around the fixture for the robot movement. The aim is that multi-level implementation of a relaxed problem makes the problem much easier to reach an intermediate state. Each state is taken individually, and generated recovery logic is applied one by one to all states. When a recovery algorithm is found that carries all of the states to a common point, this step is completed.

In the second step, after the intermediate state is obtained, a new problem of recovery procedure is defined by taking this intermediate state as the initial state and the desired state as the goal state. Constraint values are also restored to 5 mm. The cubical working envelope is reduced to half size and investigation on a second error recovery algorithm is done. Finally, two error recovery algorithms are concatenated to obtain robust recovery logic for n different number of error cases. The complete procedure is summarized in *Figure 2*.

The advantages of using a multi-level optimization procedure are as follows:

- Trying to solve a relaxed problem will eventually result in fewer iterations than the single-step optimization case because an intermediate state

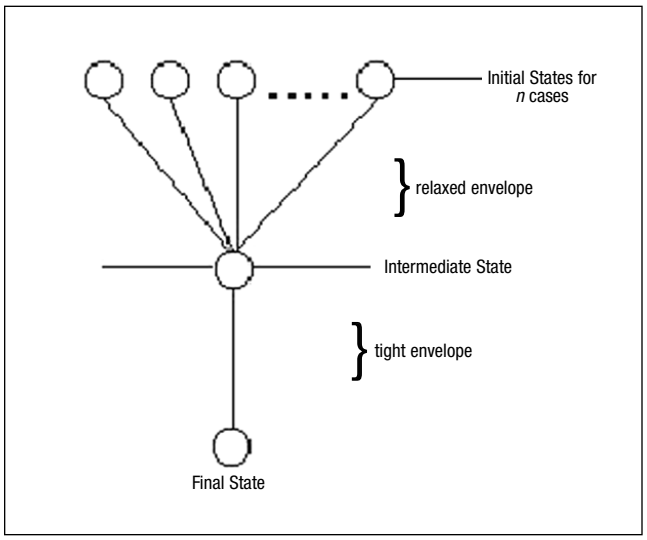

Figure 2
Multi-Level Optimization Procedure

is defined; there would be a fewer number of function evaluations.

- The recovery algorithms, which are obtained for reaching the final state from the intermediate state, can be stored as subroutines and may be used for the recovery of similar error conditions later.

A fitness function is defined individually for all error cases. These functions are taken as the inverse of the objective function as indicated through Eqs. (5) and (6), where n is the number of error cases. Therefore, the problem is converted into a maximization problem. The reason for this type of change is that generally in the usage of genetic algorithms it is preferred to have large fitness values for better members; by changing the formulation to maximization, this preference is assured.

$$i = 1, 2, ..., n \tag{5}$$

$$f_i = \frac{1}{\sqrt{(x_i - x_o)^2 + (y_i - y_o)^2 + (z_i - z_o)^2}} \tag{6}$$

The overall fitness function of a recovery program is defined as its average fitness minus the deviation between this average value and the minimum fitness value among the n cases, as is shown in Eq. (7). The variables w_1 and w_2 determine the weight of each term. In case studies, these are taken as (1,1). The purpose of defining this type of fitness function is to penalize the performance variance of the recovery program for each error case. More specifically, if a recovery algorithm performs in the same way for all error cases, its deviation is zero and the robustness is assured.

$$f = w_1 \cdot \frac{\Sigma f_i}{n} - w_2 \cdot \left(\frac{\Sigma f_i}{n} - \min f_i \right) \tag{7}$$

There can be cases where the maximum variance occurs between the average fitness and the maximum fitness. However, those types of cases are not penalized in order to keep the better solutions in the search space.

The first step of optimization is completed when a robust recovery algorithm is found. This recovery algorithm makes all of the cases reach the intermediate state. After that, the problem is renewed. The obtained intermediate state is defined as the only error case to be recovered, and the final state is taken as the goal state. At this stage, one fitness function is defined as in Eq. (8).

$$f = \frac{1}{\sqrt{(x - x_o)^2 + (y - y_o)^2 + (z - z_o)^2}} \tag{8}$$

The same procedure is applied for the second part of the problem. The obtained recovery algorithm for the second part is combined with the one obtained in the first part and robust recovery logic is obtained.

Genetic Programming Processor

In this study, during the generation of the error recovery logic genetic programming is used. The term "genetic programming" was first introduced by Koza[6] in 1992, and it addresses the problem of automatic programming, namely, the problem of how to enable a computer to do useful things by automatic programming.[7] Genetic programming uses the working principles of genetic algorithms (GAs).

Genetic algorithms were first introduced by Holland[25] in 1975. In GAs, design variables are coded onto fixed-length or variable-length strings

that are analogous to chromosomes in biological systems.[26] Strings are composed of characters, which are analogous to genes. Each string represents a solution point in the search space. An objective function is defined within the problem, and the GA tries to maximize the fitness of a solution point based on a fitness function related with the objective. In GA, two basic operators are applied to the selected pairs. The first operator is called "crossover" in which the strings of two members are cut and recombined from a random point, producing two new members. The second operator is called "mutation" and is applied by selecting a place in the string randomly and changing its value. Mutation has the advantage of introducing some diversity into the search, while crossover uses the properties of the current population to combine and produce better results.

In genetic programming, each member in the population is a computer program for the solution of the problem. The reason of selecting GP as an optimization method in this work is its power of dealing with discrete design optimization problems as well as the ease of its implementation in this case. Because an optimal recovery program is being searched, genetic programming is a perfect choice to be used in such a problem.

The commands used for this study are taken from KAREL2 language.[11] These commands enable the robot arm to approach the specified point (taking the reference as the robot base and the destination point as the tool center point) or to move relative to a predefined coordinate point. The commands used in this study are Move To, Move Away, Move Near, and Move Relative. However, to represent a blank line in the program, the Null command is also used. This command is not in the structure of the KAREL2 language, but it is implemented here to present a variable number of lines in a recovery program. The command variables are coded into a discrete set of variables, such as:

0 = Null
1 = Move To
2 = Move Away
3 = Move Near
4 = Move Relative

Each program command requires a formal way of combination with the variables of tool center point (TCP) and/or a relative movement in the negative *z*-direction (offset). *Table 1* shows the appropriate lexical representation for each command.

Table 1
Use of the Commands

COMMAND	SUFFIX
Move To	<TCP>
Move Away	<Offset>
Move Near	<TCP> By <Offset>
Move Relative	<Vector>

The TCP variable is composed of six variables. These variables are the three points in the robot frame and the three orientations of the toolholder arm. A vector is composed of three offset variables in the space. This variable coupled with the Move Relative command enables the robot to move its arm relatively from its current position in three-dimensional space.

A chromosome structure is defined for each line of a program and is composed of five genes.[27] Therefore, each program is coded into a multiple chromosome structure. The first gene in the chromosome structure is for the command value from the specified set. The second gene is the tool center point value, and it can be active depending on the command type. From the third to fifth genes, the offset values are defined. In case a vector value is needed, all of the genes from three to five are active. If the command seeks an offset value, only gene number three is active. *Figure 3* shows the chromosome structure.

The maximum number of lines for the recovery program is limited to 10. Command number 0 (Null command) represents a blank line in the program so a variable length of DNA structure is defined within the computer programs. For each chromosome, the values of the variables are taken from a set, which is randomly generated in the beginning. These values are stored in one main matrix and several submatrices in order to be retrieved and processed during the evolution process.

At first, a TCP matrix is defined and filled with the randomly generated values within the constraints. This matrix has the size of 1000 x 6. Each row represents a TCP with its position and orientation values. The maximum number of rows, 1000, is taken the multiplication of the number of programs in a population with the maximum number of lines in each program. After that, an offset value matrix is

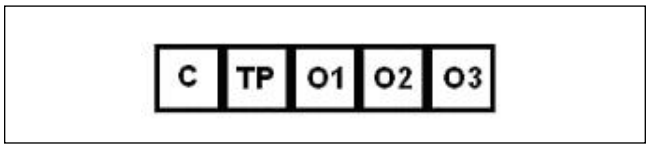

Figure 3
Chromosome Structure

defined with the size of 1000 x 1. Each row represents an offset value.

The main matrix is a three-dimensional matrix composed of all commands, teach points, and offset values and has the size of 100 x 10 x 5. Each row in the first dimension represents one computer program in the population. The second dimension is used for each line in a computer program, while the third dimension is used for the chromosome structure of each line. This matrix holds the index values from the teach points and offset matrices. These index values are combined randomly in the beginning of the optimization process and are changed according to the evolutionary rules during the process.

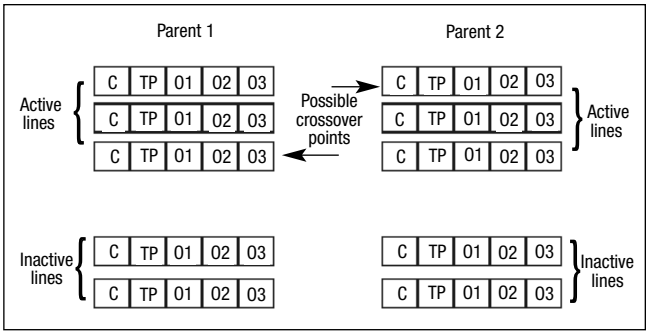

Figure 4
Determination of Crossover Line

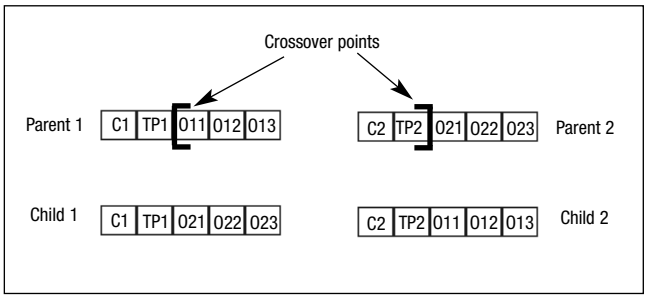

Figure 5
Chromosome Structures After Crossover

Evolution Procedure

In each generation, all of the members of the population are evaluated with respect to the desired fitness function. Both nondeterministic and deterministic types of crossover are implemented. In nondeterministic crossover, the probability is taken as 0.9. Weighted roulette wheel selection is used during the study. The probability for a member to be in the mate pool is given as:

$$p_i = \frac{f_i}{\Sigma f_i} \tag{9}$$

where f_i is the fitness value of the individual. After two parents are selected, the crossover operation takes place. The first step in the crossover process is locating the crossover line point between the programs. Each program has a different number of lines (that is, a variable number of DNA strands). However, due to the structure of the program, it is possible to not have all the lines "active". If a collision has occurred during the execution of the program and automatic recovery from this collision is not achievable by using the rest of the command lines, the rest of the program becomes "inactive". These inactive lines are not pruned because they may become active after the mutation. Therefore, a variable, which keeps the number of working lines for each program, is stored. When crossover takes place, the random crossover point selection is chosen from the active lines. *Figure 4* depicts the action.

After the crossover line is determined, the next step is locating the crossover point on the chromosome structure. A point on the chromosome is selected randomly, and two children are generated in the next generation by exchanging the genes after the crossover point. *Figure 5* shows the crossover operation.

The second type of crossover uses deterministic rules. At first, each member is ranked based on its performance for each error state. After that, the inactive lines of each program are pruned and their active lines are combined, suggesting that this kind of elitism produces better results. For n different states, this operation produces e number of programs given with the following equation. The (n,k) function represents the combination of n in k different ways:

$$e = \sum_{k=2}^{n} \binom{n}{k} \tag{10}$$

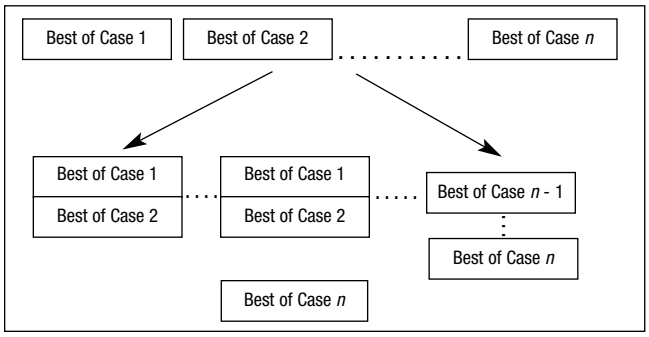

Figure 6
Deterministic Crossover Procedure

Figure 6 demonstrates the overall operation. It is observed that this type of elitism improves the evolution process.

Mutation is not applied to this part of the crossover. After programs are combined in this manner, they are directly inserted to the next generation. The rest of the members for the next generation are obtained from the probabilistic crossover part.

Mutation takes place after the probabilistic crossover operation to prevent premature convergence. By applying mutation, a small portion of new members is introduced to the population. In this study, variable mutation probability is implemented. The bias of mutation is focused on the active lines but not limited to inactive ones. This type of mutation may turn the inactive lines in the chromosome structure to become active. During the operation, a mutation line is selected first randomly. After that, in this line a mutation point is determined again randomly.

After the evolution step, the new generation of the programs is stored in the main matrix structure. This encoded structure is decoded into working programs in KAREL2 language. These programs are tested in Workspace and their outputs are written into text files. After that, these outputs are processed with the developed program and the next evolutionary step takes place according to these outputs. The summary of the developed system is given in *Figure 7*.

The implemented system is tested on several case studies. The results demonstrated that the system's overall performance is efficient to find robust recovery logic. These experimental results are given in the next section.

Case Studies

A model assembly line is constructed by using Workspace simulation software, and the details are given in Baydar and Saitou.[3,24] An IRB6000 type industrial robot is used for the part placement procedure during the assembly process.

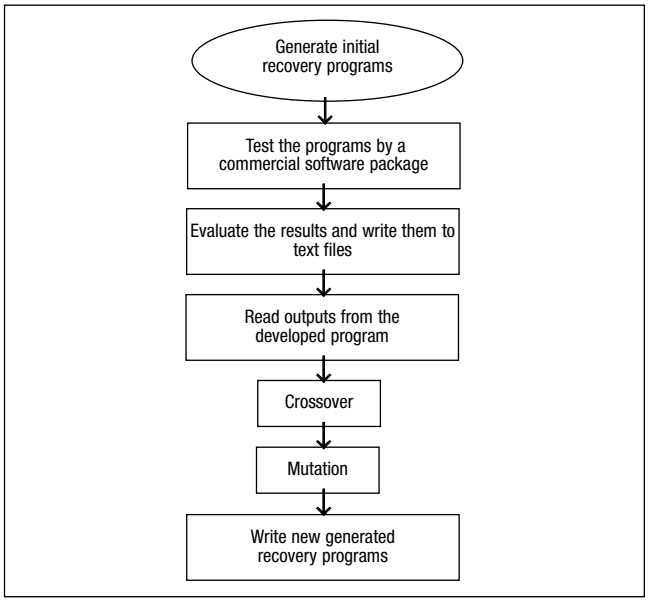

Figure 7
Working Mechanism of Developed Framework

In the following case studies, at first only one collision case, which is between the workpiece and the fixture, is considered to recover. Then multi-level optimization of three different collision points is presented. In the case studies, the assumption is made that the part is held in the gripper after the collision and that repositioning can be detected properly. *Figure 8* shows the model of the assembly line and the desired position of the workpiece on the fixture.

Case Study 1

The first case study is taken from a collision state occurring between the workpiece and the fixture, as shown in *Figure 9*. Because this problem includes only one state to recover, multi-level optimization and deterministic crossover features are not used. Several runs are carried out with the developed infrastructure. A cubical space for the robot movement is defined as the working envelope, which has the size of 400 x 400 x 400 mm^3. As is observed from the outputs (*Table 2*), an error reduction of 65% is gained in the replacement of the workpiece after the initial generation. In generation 11, an optimal result is found. This is a local optimum; however, it is acceptable because it is in the tolerance limits.

The best program obtained after the 11th generation is composed of the lines stated below. The last

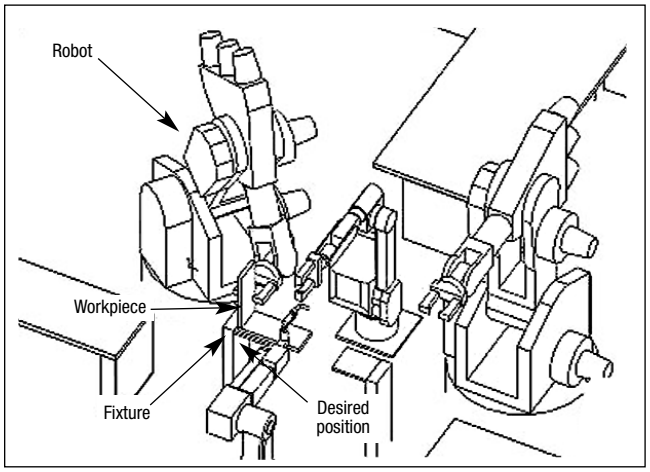

Figure 8
Modeled Assembly Line with Workpiece at Desired Position

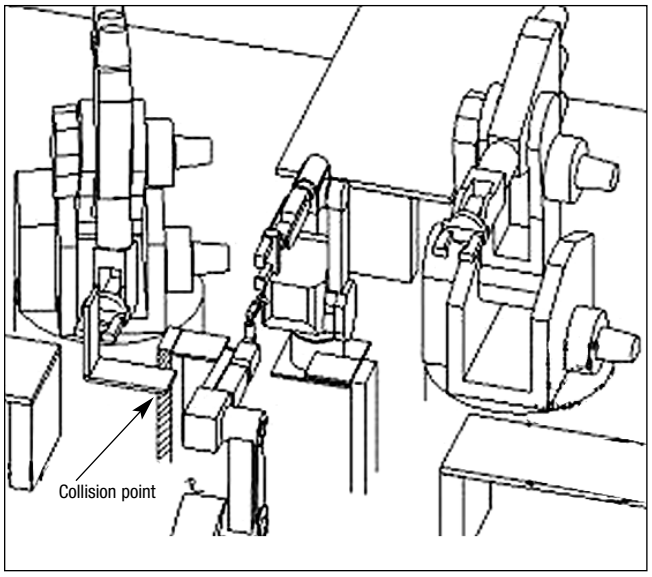

Figure 9
Collision Case Between Workpiece and Fixture

command line is added automatically by the system to restore the desired final orientation.

```
' Best Program of Case Study 1:
ROUTINE GPCode26
BEGIN
Move To POS (-710, -684, -982, 90, 80, 0,'RUFB')
Move Relative (9,8,10)
Move To POS (-701 -692, -992, 90, 90, 0,'RUFB')
END GPCode26
```

During the recovery process, it is observed that the algorithm is composed of three lines only between the 'BEGIN' and 'END" commands. The maximum lines is limited to 10, and this result shows that recovery is accomplished in a significantly small number of steps. The resulting placement errors are given in *Table 3*.

Table 2
Outputs from Case Study 1

Generation	Objective Function
1	24.535
2	31.968
3	14.866
4	14.866
5	14.212
6	13.967
7	12.180
8	12.180
9	11.682
10	9.708
11	8.66

Table 3
Placement Errors

Coordinate	Error (mm)
X	5
Y	5
Z	5

The history of the optimization is given in *Figure 10*. It is observed that both the fitness values of the best and the worst recovery programs are increasing as the evolution takes place. It should be noted that the fitness function (to be maximized) is the inverse of the objective function (to be minimized).

Case Study 2

Three collision points are studied in this case to find a robust recovery algorithm. The first error state is taken from the previous case study. *Figures 11* and *12* show the other collision points. Because there are multiple error states to recover, multi-level optimization and deterministic crossover features are used in this case. At first, the first level of optimization is accomplished. This is completed in 10 generations (counting the initial random generation as the first generation).

The positional errors after the 10th generation at the intermediate state are given in *Table 4*. Note that these values are obtained with the relaxed constraints.

After reaching the intermediate state, the second level of optimization is started by regenerating the population. After the seventh generation, a local

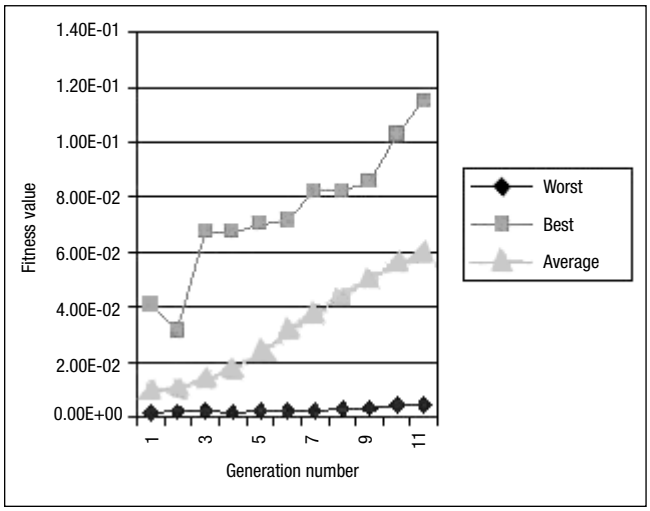

Figure 10
Optimization History of Case Study 1

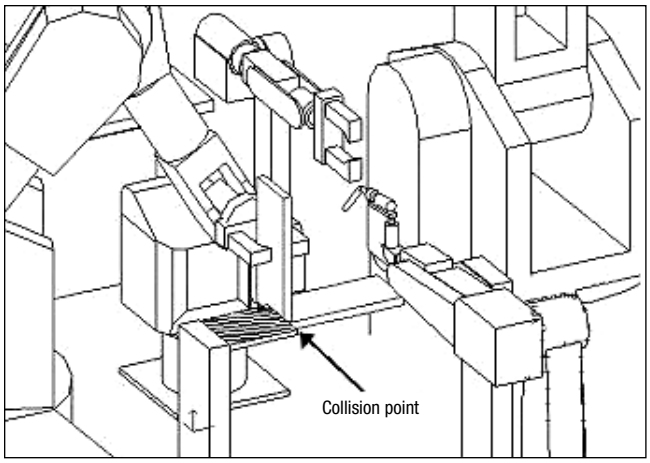

Figure 11
Second Collision Point in Case Study 2

optimum is found. *Table 5* shows the final placement errors for the final state. Note that in this case original constraint values are restored.

In total, 17 generations are needed to reach the robust recovery algorithm, and it is composed of six lines between the BEGIN and END command. For the first stage of the optimization, the recovery algorithm contains two lines of code. In the second stage, four additional lines are added to the code. The recovery algorithm is given below.

```
ROUTINE RecoveryCase2
BEGIN
— This portion of the code below belongs to the first level
Move To POS( -798, -794, -1029, 50, 50, 0,'RUFB')
Move To POS( -704, -708, -970, 50, 80, 0,'RUFB')
—This portion of the code below belongs to the second level
Move To POS( -718, -661, -1028, 40, 10, 0,'RUFB')
```

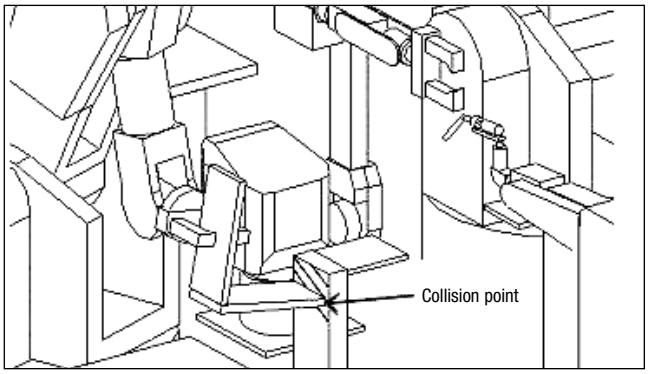

Figure 12
Third Collision Point in Case Study 2

Table 4
Positional Errors at Intermediate State

Coordinate	Error (mm)
X	2
Y	11
Z	15

Table 5
Positional Errors at the Final State

Coordinate	Error (mm)
X	5
Y	3
Z	3

```
Move Away -51
Move To POS( -718, -661, -1028, 40, 10, 0,'RUFB')
Move To POS( -711, -694, -990, 90, 90, 0,'RUFB')
End RecoveryCase2
```

Table 6 gives the history of the objective function. It is observed that a fluctuation occurred between the 10th and 11th generations. The reason is that the second stage of the optimization is initiated at the 11th generation with a new generation of population. This can be seen from *Figure 13* also. Note that the fitness function is inversely proportional with the objective function; therefore, it is increasing throughout the study.

The performance of the robust recovery algorithm is tested on each error case individually, and it is found that the procedure is working properly. The obtained recovery code of the first case study is also tested in this case study, but the results are not found to be acceptable.

As experienced in the second case study for three states, it took 17 generations to reach an acceptable optimum within the acceptable tolerance limits; however, in the first case study for one state, it took

Table 6
Change in the Objective Function

Generation	Objective Function
1	56.311
2	37.23
3	37.23
4	36.959
5	36.969
6	33.030
7	33.030
8	32.465
9	32.465
10	20.346
11	24.39
12	15.78
13	15.556
14	15.556
15	15.556
16	15.556
17	6.556

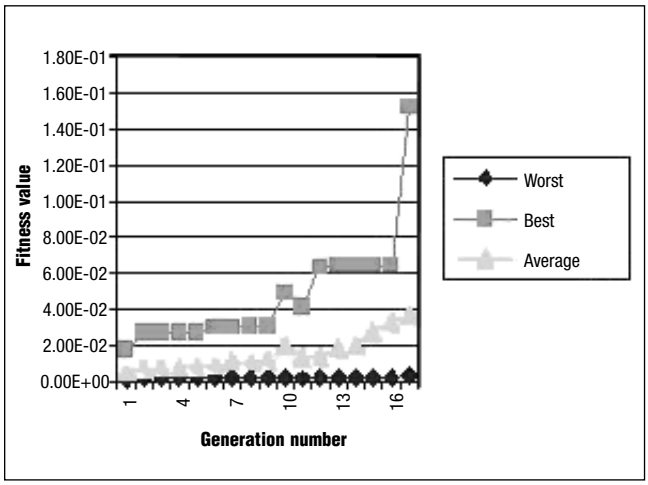

Figure 13
Optimization Progress of Case Study 2

11 generations. Besides, the final positioning of the second case study is better than the one in the first case study. It is experimented that the deterministic crossover and multi-level optimization procedures improved the convergence. It is also observed that the length of the recovery program is dependent on the initial number of states and their positions in the space.

Case Study 3

A different set of three points is selected at this time to test the performance of the system. These three collision positions are given in *Figures 14-16*. This time it took seven generations to reach the intermediate state. The positional error in each dimension at the intermediate state is given in *Table 7* below.

After this point, second level of optimization is initiated. The second state reaches the limits of the final state in four generations. The final placement errors are given in *Table 8*.

In total, 11 generations are required to gather the robust recovery code. Two recovery algorithms are combined, and a robust recovery for these three cases is obtained. The final algorithm is composed of five lines between the BEGIN and END commands, as given below. The first two lines are from initial state to intermediate state, while the rest of the lines are for the recovery from intermediate state to final state. The change in the objective function is given in *Table 9*.

```
ROUTINE Recovery2
BEGIN
— This portion of the code below belongs to the first level
Move Relative VEC (0, -70, -8)
Move Near POS (-702, -655, -1006, 50, 10, 0,'RUFB') By -43
—This portion of the code below belongs to the second level
Move Near POS (-733, -730, -1005, 80, 20, 0,'RUFB') By -74
Move Near POS (-713, -688, -991, 70, 20, 0,'RUFB') By -15
Move To POS (-711, -702, -986, 90, 90, 0,'RUFB')
END Recovery2
```

The change in the objective function from intermediate state to final state is smooth in this case. However, as can be seen from *Figure 17*, the average fitness of the members dropped at the eighth generation because a new population is generated for the second-level optimization process. It is realized that the average value increased between the eighth and 10th generations and this helped the best fitness increase in the 11th generation.

It is noted that the second level of optimization converged in four generations in this case. During the case studies, three error states are considered for each case study. However, the number of cases in the initial set can be increased to obtain robust recovery logic for *n* number of cases.

Collision detection between the objects is handled internally by the software package. This detection consumes minimal time during the iterative solution-finding process and is dependent on the CPU used. Evaluation of each generation takes around three minutes on a PIII-450 CPU during the case studies.

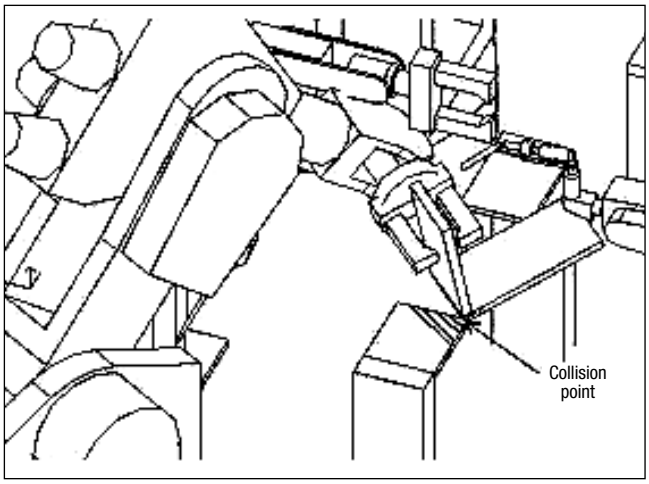

Figure 14
First Collision Point in Case Study 3

Table 7
Positional Errors at the Intermediate State

Coordinate	Error (mm)
X	9
Y	9
Z	8

Table 8
Positional Errors at Final State

Coordinate	Error (mm)
X	5
Y	5
Z	1

Table 9
Change in the Objective Function

Generation	Objective Function
1	38.105
2	24.35
3	24.35
4	18.815
5	18.815
6	16.301
7	15.033
8	12.083
9	12.083
10	12.083
11	7.1414

Discussion and Conclusions

Automated assembly lines are subject to unexpected failures during their normal operation. Such failures may lead to costly shutdowns and generally require on-line diagnosis and recovery by human experts to restore their normal operation. Previous

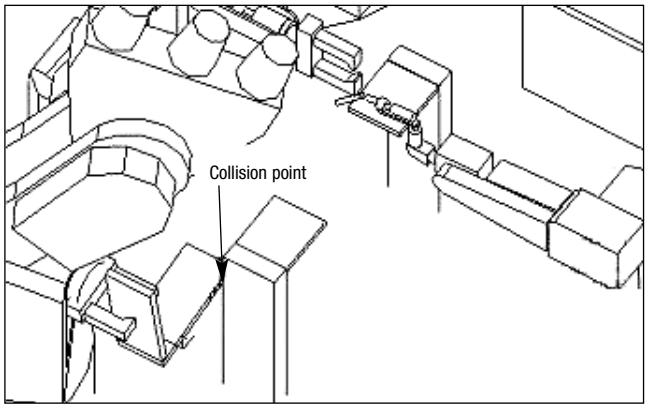

Figure 15
Second Collision Point in Case Study 3

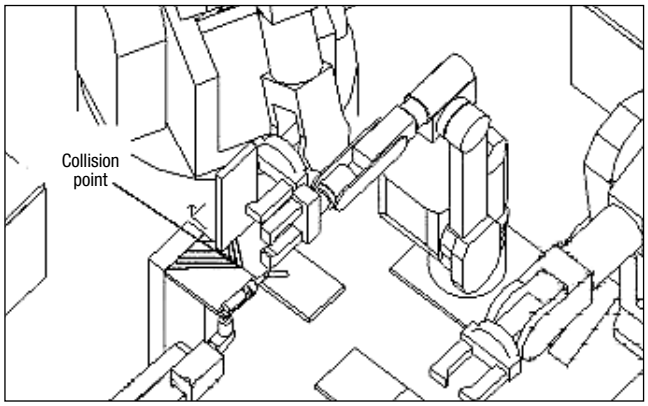

Figure 16
Third Collision Point in Case Study 3

methods for automated recovery are focused on the on-line recovery and training of the assembly lines using planners based on expert systems. However, these planner systems are aimed at developing a plan for the anticipated error scenarios and, after that, converting the generated plan to an understandable controller code. However, this type of approach requires the post-processing of the plan and is based on expected error situations.

In this paper, a different approach, which uses three-dimensionally modeled systems, is proposed. A computer program is developed and coupled with a commercial software package to work off-line on the robust error recovery logic generation by using genetic programming. The proposed approach uses a three-dimensional model of the assembly system to generate and test robust recovery logic for the given error conditions. The use of genetic programming enables recovery code to be produced in the controller language itself. Therefore, the need for post-processing is eliminated. Besides, the system is cou-

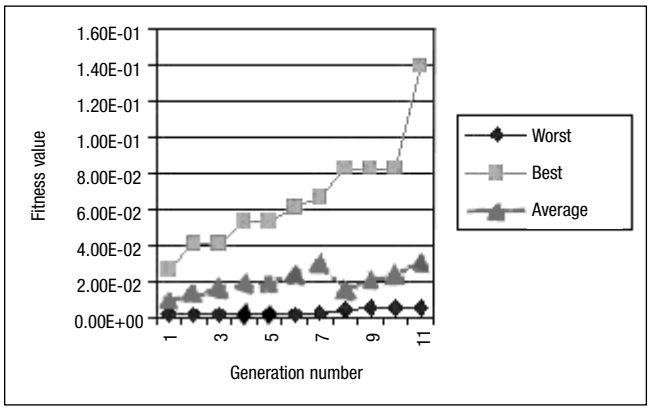

Figure 17
Optimization History of Case Study 3

pled with 3-D robotic simulation software to generate unexpected error states, which previous approaches are deficient to fulfill. The system uses deterministic and nondeterministic crossover and variable mutation schemes with the implementation of a multi-level optimization procedure to improve the performance.

Because part misplacement errors widely occur during the assembly process, recovery for collision errors is studied. However, the system is not limited to the collision errors only, and other error types can be studied to recover in the future. The recovery logic generation procedure accomplishes the multi-level optimization process in two steps. In the first step, several error states are studied in parallel to find a common recovery algorithm for the relaxed program. After that, in the second stage the solution of the relaxed problem is taken as the only state to be recovered for the original problem. Finally, the recovery algorithms for both stages are combined to get a robust recovery algorithm. It is observed that this procedure:

- Simplifies the problem of solving different error states. Trying to solve a relaxed problem will eventually result in fewer iterations than the single-step optimization case. Therefore, the computation speed would be increased.
- Leads to a robust error recovery algorithm that can be used for different error states (that is, collision errors occurred at different points in an assembly line).

Although three states are considered in case studies, this procedure can be applied to a larger number of states. Due to its geometrical features, each assembly line has different number of critical states to be considered. Therefore, error sampling by using the statistical model of the dimensional and functional errors must be investigated for each line to find robust recovery algorithms for each error case. Future studies are aimed at this type of error case analysis by using the Monte Carlo method.

Several limitations are identified within this methodology. Currently, a recovery logic code containing commands for one robot is being generated. However, in some cases multi-actuator commands (that is, controlling a turntable along with a robot) may be necessary to use more than one agent for the recovery process. Also, because of the software limitations the dynamics of the parts (part slippage after collision, part deflection, and so on) was not implemented. However, possible part slippages or deflections can be obtained and used as the initial error states by using a more capable software package that realistically simulates dynamics. After performing statistical process capability, the possible slippage and deflection states can be obtained and a robust recovery with the proposed approach can be found no matter how the slippage occurs.

Performance of the system is evaluated with the simulations, and the results showed that the deterministic crossover and multi-level optimization features improved the progress and that the system is efficient to solve recovery problems and capable of generating robust recovery plans for multiple different error states. It is expected that this approach will require less time for the generation of error recovery logic.

References

1. J.T. Luxhoj, J.O. Riis, and U. Thorsteinsson, "Trends and Perspectives in Industrial Maintenance Management," *Journal of Mfg. Systems* (v16, n6, 1997), pp437-453.

2. M.C. Zhou and F. DiCesare, "Adaptive Design of Petri-Net Controllers for Error Recovery in Automated Manufacturing Systems," *IEEE Trans. on Systems, Man and Cybernetics* (v19, n5, 1989), pp963-973.

3. C. Baydar and K. Saitou, "Off-Line Error Recovery Logic Synthesis in Automated Assembly Lines by Using Genetic Programming," 2000 Japan-USA Symp. on Flexible Automation (2000).

4. M.L. Visinsky, J.R. Cavallaro, and I.D. Walker, "Expert System Framework for Fault Detection and Fault Tolerance in Robotics," *Computers in Electrical Engg.* (v20, n5, 1994), pp421-435.

5. L.S. Lopes and L.M. Camarinha-Matos, "Learning to Diagnose Failures of Assembly Tasks," *Artificial Intelligence in Real Time* (1994), pp97-103.

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

7. W. Banzhaf, P. Nordin, R. Keller, and F. Francone, *Genetic Programming: An Introduction* (Morgan Kauffman Publishers, 1998), pp4-5.
8. L. Whitcomb and D.E. Koditschek, "Automatic Assembly Planning and Control via Potential Functions," *Proc.* of IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems—IROS '91, Osaka, Japan (1991), pp17-23.
9. R.A. Jarvis, "Projection Derived Space Cube Scene Models for Robotic Vision and Collision-Free Trajectory Planning," Robotics Research, 2nd Int'l Symp., Kyoto, Japan, 1985 (Cambridge, MA: MIT Press, 1985), pp403-410.
10. P. Pal and K. Jayarajan, "Fast Path Planning for Robot Manipulators Using Spatial Relations in the Configuration Space," *Proc.* of IEEE Int'l Conf. on Robotics and Automation (v2, 1993), pp668-673.
11. Flow Software Inc., Workspace Educational User Guide Manual (v4, 1998).
12. S. Srinivas, "Error Recovery in Robot Systems," PhD thesis (California Institute of Technology, 1977).
13. S. Brnyjolfsson and A. Arnstrom, "Error Detection and Recovery in Flexible Assembly Systems," *Int'l Journal of Advanced Mfg. Technology* (v5, 1990), pp112-125.
14. M.G. Abu-Hamdan and A.S. El-Gizawy, "Computer Aided Monitoring System for Flexible Assembly Operations," *Computers in Industry* (v34, 1997), pp1-10.
15. S.G. Tzafestas and G.B. Stamou, "Concerning Automated Assembly: Knowledge-Based Issues and a Fuzzy System for Assembly under Uncertainty," *Computer Integrated Mfg. Systems* (v10, n3, 1997), pp183-192.
16. S.M. Telagi and A.H. Soni, "Control Systems in Manufacturing," *Proc.* of 1994 ASME Design Technical Conf.—4th ASME Flexible Assembly Conf. (v73, 1994), pp17-23.
17. E.Z. Evans and S.G. Lee, "Automatic Generation of Error Recovery Knowledge Through Learned Activity," *Proc.* of 1994 IEEE Int'l Conf. on Robotics and Automation (v4, 1994), pp2915-2920.
18. J.F. Kao, "Optimal Recovery Strategies for Manufacturing Systems," *European Journal of Operations Research* (v80, 1995), pp252-263.
19. T.C. Cao and A.C. Sanderson, "Sensor-Based Error Recovery for Robotic Task Sequences Using Fuzzy Petri-Nets," *Proc.* of 1992 IEEE Int'l Conf. on Robotics and Automation (v2, 1992), pp1063-1069.
20. K. Khodabandehloo, "Analyses of Robot Systems Using Fault and Event Trees: Case Studies," *Reliability Engg. and System Safety* (v 53, 1996), pp247-264.
21. N. Hardy, D. Barnes, and M. Lee, "Automatic Diagnosis of Task Faults in Flexible Manufacturing Systems," *Robotica* (v7, 1989), pp25-35.
22. T. Kis and J. Vãncza, "Computational Complexity of Manufacturing Process Planning," New Directions in AI Planning: EWSP '95—3rd European Workshop on Planning, Frontiers in AI and Applications, Assisi, Italy, 1994, pp299-311.
23. I. Klein, P. Jonsson, and C. Backstrom, "Efficient Planning for a Miniature Assembly Line," *Artificial Intelligence in Engg.* (v13, 1999), pp69-81.
24. C. Baydar and K. Saitou, "Generation of Robust Error Recovery Logic in Assembly Systems Using Multi-Level Optimization and Genetic Programming," ASME/DETC 2000 Computers in Engg. Conf. (2000).
25. J. Holland, *Adaptation in Natural and Artificial Systems* (Cambridge, MA: MIT Press, 1975).
26. D. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley, 1989).
27. C. Baydar and K. Saitou, "A Genetic Programming Framework for Error Recovery in Assembly Systems," Genetic and Evolutionary Computation 2000 Conf. (2000).

Authors' Biographies

Cem Baydar is a graduate student seeking his PhD in the Dept. of Mechanical Engineering and Applied Mechanics at The University of Michigan. He received his MSc degree in mechanical engineering from the Middle East Technical University in Turkey in May 1998. His research interests include design optimization, mechatronics, and Internet-based distance engineering design. Mr. Baydar is a member of ASME and IEEE.

Kazuhiro Saitou is an assistant professor in the Dept. of Mechanical Engineering and Applied Mechanics at The University of Michigan. Dr. Saitou received his PhD in mechanical engineering from the Massachusetts Institute of Technology in June 1996. His research interests include design automation and optimization, design for manufacturing and manufacturing system design, micro assembly, and evolutionary computation in design. He received a 1999-2000 NSF CAREER Award. Dr. Saitou is a member of ASME, ASEE, and IEEE.