Proceedings of DETC'00
ASME 2000 Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
Baltimore, Maryland, September 10-13

DETC2000/CIE-14639

GENERATION OF ROBUST ERROR RECOVERY LOGIC IN ASSEMBLY SYSTEMS USING MULTI-LEVEL OPTIMIZATION AND GENETIC PROGRAMMING

Cem M. BAYDAR
(cbaydar@umich.edu)

Kazuhiro SAITOU*
(kazu@umich.edu)

Department of Mechanical Engineering and Applied Mechanics
University of Michigan
Ann Arbor, MI 48109-2125, USA

ABSTRACT

Automated assembly lines are subject to unexpected failures, which can cause costly shutdowns. Generally, these errors are handled by human experts or logic controllers. However, these controller codes are based on anticipated error scenarios and are deficient in dealing with unforeseen situations. In our previous work (Baydar and Saitou, 2000a), an approach for the automated generation of error recovery logic was discussed. The method is based on three-dimensional geometric modeling of the assembly line to generate error recovery logic in an "off-line" manner using Genetic Programming. The scope of our previous work was focused on finding an error recovery algorithm from a predefined error case. However due to the geometrical features of the assembly lines, there may be cases which can be detected as the same type of error by the sensors. Therefore robustness must be assured in the sense of having a common recovery algorithm for similar cases during the recovery sequence. In this paper, an extension of our previous study is presented to overcome this problem. An assembly line is modeled and from the given error cases optimum way of error recovery is investigated using multi-level optimization. The obtained results showed that the infrastructure is capable of finding robust error recovery algorithms and multi-level optimization procedure improved the process. It is expected that the results of this study will be combined with the automatic error generation, resulting in efficient ways to automated error recovery logic synthesis.

INTRODUCTION

Error recovery plays an important role in automated assembly systems since these systems are open to unexpected failures, which can halt their operation. Generally, recovery algorithms for such failures are anticipated by *on-line* investigation of the assembly line by the experts, during the design of assembly lines. Another approach is using Programmable Logic Controller (PLC) codes, which are also manually coded, based on anticipated scenarios. However prediction of all scenarios is impossible, therefore these methods are not flexible to solve the majority of the problems. An approach of using *off-line* synthesis of error diagnosis and recovery logic based on the three-dimensional geometry model of an entire assembly line was discussed in our previous work (Baydar and Saitou, 2000a). The scope of this work was finding an error recovery algorithm from a predefined error case. The system uses one of the commercial assembly line simulation software (Workspace, 1998), which is coupled with a developed computer program, to obtain error recovery logic using Genetic Programming (Koza, 1992). Previous results showed that, the system is capable of generating recovery logic for collision errors from various error cases individually. However it does not provide robust recovery logic for multiple error cases. The work discussed in this paper is an extension of our previous study and aimed to recover this deficiency.

The following section contains information on the previous work done on error diagnosis, highlighting the importance of having a robust recovery logic as well as a brief summary on our previous approach.

* Corresponding author

PREVIOUS WORK

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 (Khodabandehloo, 1997). 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 developed a knowledge-based system for monitoring, diagnosis and error recovery for the flexible assembly operations (Abu-Hamdan and El-Gizawy, 1994). 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 sub-goals of the tree. The facts of the errors (i.e. sensor failure) are represented as the leaves of the sub-goals. The use of fault trees as a database of run-time fault detection is discussed in (Visinsky *et al.*, 1994). 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. Other proposed two methods are known as Failure Reason Analysis (FRA) and Multiple Outcome Analysis (MOA), which are discussed in (Hardy *et al.*, 1989). 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 passes 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.

All of the systems discussed above are focused on diagnosis and recovery by using expected error cases. However, due to the geometrical nature of the assembly lines, there can be errors, which have the same error type (i.e. collision) but need to be recovered by using a procedure different from the anticipated case. For example a collision error can be occurred 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.

In our previous study (Baydar and Saitou, 2000a), a possible way of error recovery in *off-line* manner was discussed. A sample assembly line was modeled three-dimensionally using a commercial software package.

The architecture of the system is summarized in Figure 1. The system uses a commercial software package called Workspace (Workspace, 1998). A software module was developed and coupled with the Workspace. This module is responsible from the generation of recovery logic using Genetic Programming (Koza, 1992). The generated programs are tested with the commercial software and evaluated based on their performance. After that, evolution process takes place according to the performance of each program. Case studies demonstrated that the developed system is efficient to find the optimum recovery logic from a given collision case. For the error recovery language, KAREL2 Robot Language was selected. The commands for this language are used to manipulate the robot for the recovery process.

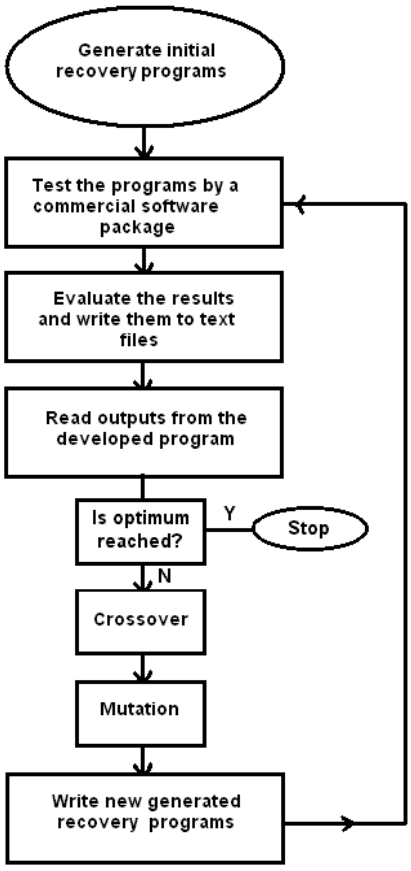

Figure.1: System Architecture.

The main disadvantage of the previous system is due to its insufficiency to generate robust recovery logic from multiple error cases. As it is stated before, a part presentation error can be resulted in a collision error in different ways at different places in three-dimensional space during the assembly process. Therefore, a robust recovery algorithm should be investigated for the recovery from different collision cases.

PROPOSED APPROACH

Genetic programming is an extension of genetic algorithms, aimed to produce useful computer programs automatically to solve a specific problem. The term "Genetic Programming" was first introduced by Koza (Koza, 1992). It uses the same working principles of Genetic algorithms.

Genetic algorithms were first introduced by Holland (Holland, 1975). They borrow their terminology and working principles from the biology. Basically, problem variables are coded onto strings like chromosomes in biological systems. Each string is a member of the population, representing a solution for the problem. The aim is maximizing a fitness function based on the defined objective of the problem. Two basic operators are used for the evolution process. The first operator, which is called "crossover", is responsible for taking two strings as parents and combining them to produce better strings as children. The second operator is called "mutation" which has the advantage of introducing some diversity to the population by changing the values in a string randomly.

Genetic programming requires efficient representation of the search space with the definition of several critical variables for the population. As in our previous study, a chromosome structure is defined for each line within a recovery algorithm. The maximum number of lines is limited to 10. The number of members in the population is determined as 100. The crossover probability is taken as 0.9 and it is directly proportional with the fitness value of the recovery algorithm. Two parents are selected based on their fitness value and the crossover operation takes place by locating a crossover point between the programs. In order to introduce variance to the population, dynamic mutation was applied with a probability of changing between 0.015 and 0.05 depending on the nature of the population.

During this study as in our previous work part placement errors, which result in collision, are studied. Collision calculations are performed by using the commercial software package's abilities. 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. Therefore the problem is a single objective optimization problem with the objective function:

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

During the recovery procedure, tolerances for the final position of the workpiece are determined as 5 mm for a successful assembly operation in all dimensions.

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

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

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

The problem is handled in two phases:

- 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.

At first, 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 this time the constraints are relaxed from 5 mm. to 15 mm. The reason for relaxing the constraints to 15 mm is to find a common point near to the desired location. A feasible working envelope is defined around the fixture for the robot movement. It is aimed that this kind of multi-level implementation makes the problem much easier to reach an intermediate state when the problem is relaxed.

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. 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.

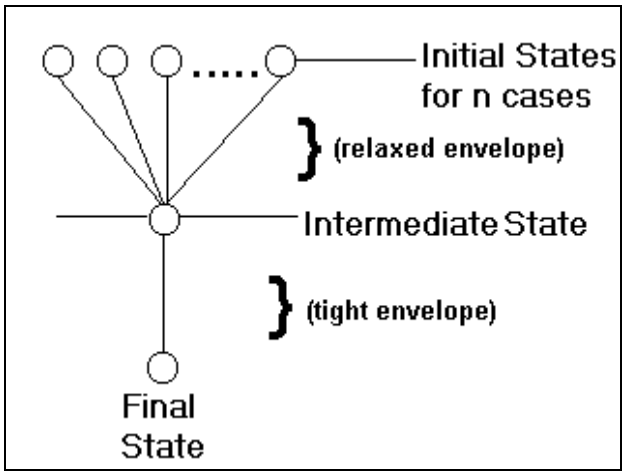

Figure.2: Multi-level optimization Procedure.

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

- Trying to solve a relaxed problem will eventually result in less number of iterations than the single-step optimization case since an intermediate state is defined; there would be less number of function evaluations.

- The recovery algorithms, which are obtained for reaching the final state from the intermediate state can be stored as sub-routines 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 Eq. (5) and (6) where n is the number of error cases. Therefore the problem is converted into a maximization problem.

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

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

The overall fitness of a recovery program is defined as its average fitness minus the absolute value of the deviation between this average value and the minimum fitness value among the n cases as it is shown in Eq. (7). The variables w_1 and w_2 determine the weight of each term. By using this kind of definition, performance variance for recovery program is penalized and robustness is tried to be assured.

$$f = w_1 \cdot \frac{\sum f_i}{n} - w_2 \cdot ABS \mid \frac{\sum f_i}{n} - \min f_i \mid \tag{7}$$

There can be cases where the maximum variance is occurred between the average fitness and the maximum fitness. However, those types of cases are not penalized to keep the better solutions in the search space. Detailed information on the evolutionary coding of the problem is given in (Baydar and Saitou, 2000b).

The first step of optimization is completed when a robust recovery algorithm is found. This recovery algorithm makes all of the cases to 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 such as in Eq. (8).

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

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.

In the Genetic Programming part, same population structure from our previous study is used. As an improvement from our previous study, during the evolution process *elitism* (Goldberg, 1989) is also introduced. The recovery programs are ranked based on their individual case performance. After that, best programs for each case are combined to get better solutions in the problem domain. The case studies showed that this type of implementation increased the performance of the system. It was observed that, there could be recovery programs, which perform efficiently for only small portion of the error cases and combination of these programs would result in better programs for generation of the robust recovery.

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.

CASE STUDIES

A model assembly line is constructed by using Workspace simulation software and the details are given in (Baydar and Saitou, 2000a). An IRB6000 type industrial robot is used for the part placement procedure during the assembly process. Figure.3 shows the model of the assembly line and the desired position of the workpiece on the fixture.

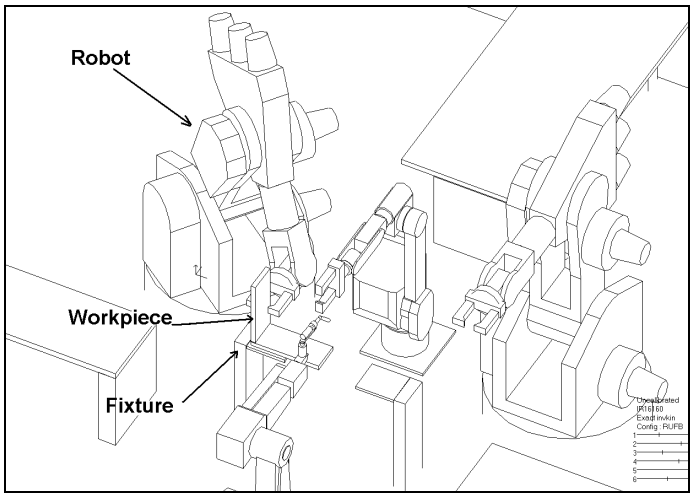

Figure.3: Modeled Assembly Line.

In the following case studies, six different collision points are generated randomly between the workpiece and the fixture. These six error cases are studied in two different case studies. In case studies, the assumption is made on that the part is held in the gripper after the collision and repositioning can be detected properly.

Case Study 1:

Three collision points (n=3) are studied to find a robust recovery algorithm. Two of these points are taken from our previous study. Figures 4-6 show those collision points. At first the first level of optimization is accomplished. This is completed in 10 generations (counting the initial random generation as the first generation).

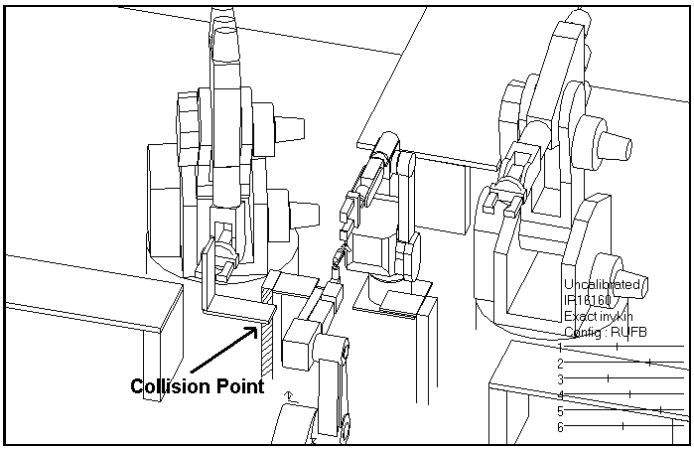

Figure.4: 1st Collision Point in Case Study 1.

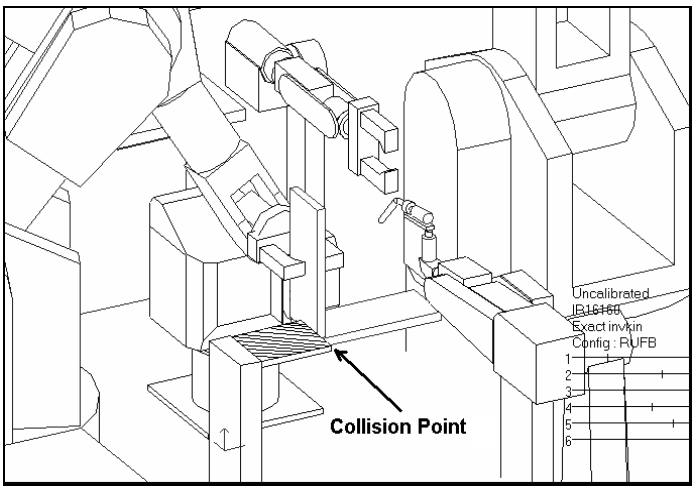

Figure.5: 2nd Collision Point in Case Study 1.

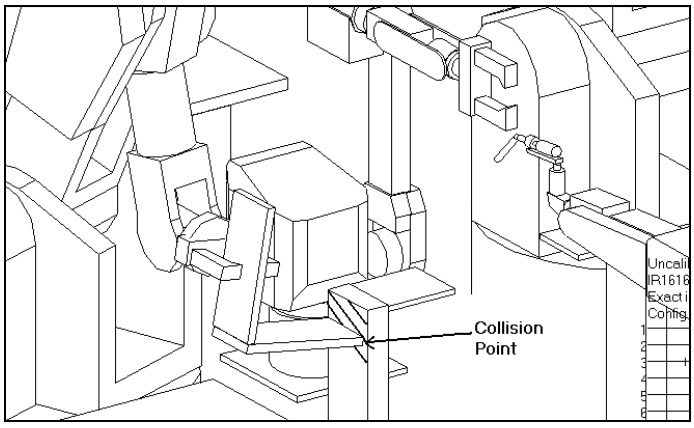

Figure.6: 3rd Collision Point in Case Study 1.

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

Table.1: Positional Errors at the Intermediate State.

Coordinate:	Error (mm):
X:	2
Y:	11
Z:	15

After reaching the intermediate state, second level of optimization is started by regenerating the population. After the 7th generation a local optimum is found. Table.2 shows the final placement errors for the final state. Note that in this case original constraint values are restored.

Table.2: Positional Errors at the Final State.

Coordinate:	Error (mm):
X:	5
Y:	3
Z:	3

Totally, 17 generations are needed to reach the robust recovery algorithm and it is composed of 6 lines between the BEGIN and END command. For the first stage of the optimization the recovery algorithm contains 2 lines of code. In the second stage, 4 additional lines are added to the code. The recovery algorithm is given below.

```
ROUTINE RecoveryCase1
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')
Move Away -51
Move To POS( -718, -661, -1028, 40, 10, 0,'RUFB')
Move To POS( -711, -694, -990, 90, 90, 0,'RUFB')
End RecoveryCase1
```

In Table 3 the history of the objective function is given. It is observed that a fluctuation occurred between the 10th and 11th generation. 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.7 also. Note that the fitness function is inversely proportional with the objective function therefore it is increasing throughout the study.

Table.3: 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

The performance of the robust recovery algorithm is tested on each error case individually and it is found that the procedure is working properly.

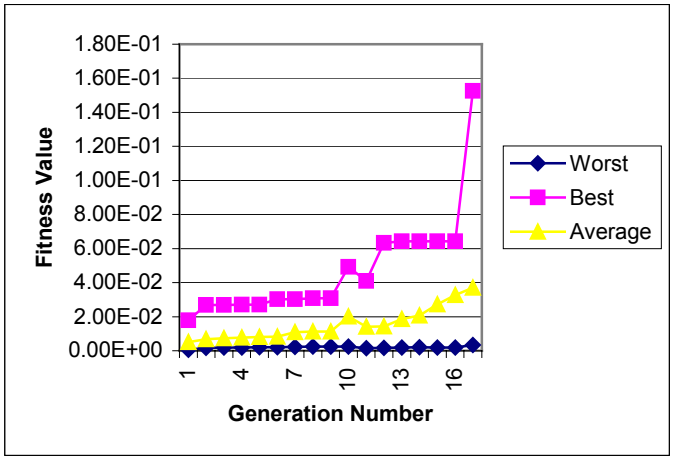

Figure.7: Optimization Progress of the 1st Case Study.

Case Study 2:

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 8-10. This time it took 7 generations to reach the intermediate state. The positional error in each dimension at the intermediate state is given in the Table.4 below.

Table.4: Positional Errors at the Intermediate State.

Coordinate:	Error (mm):
X:	9
Y:	9
Z:	8

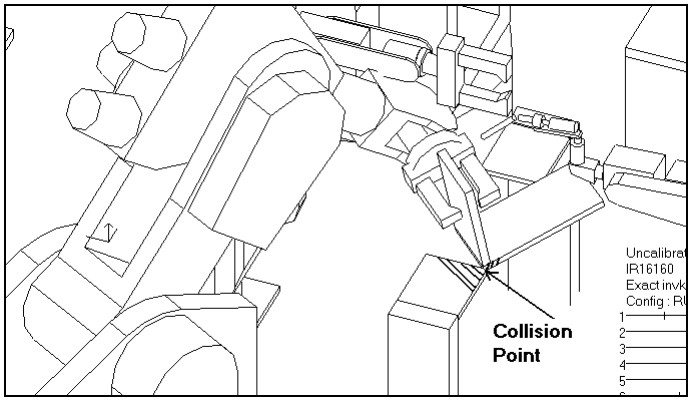

Figure.8: 1st Collision Point in Case Study 2.

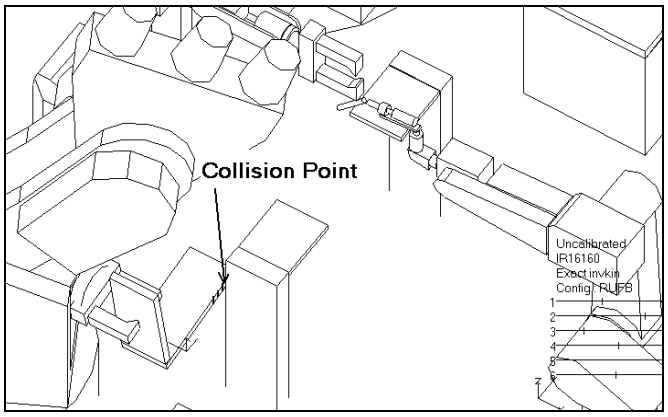

Figure.9: 2nd Collision Point in Case Study 2.

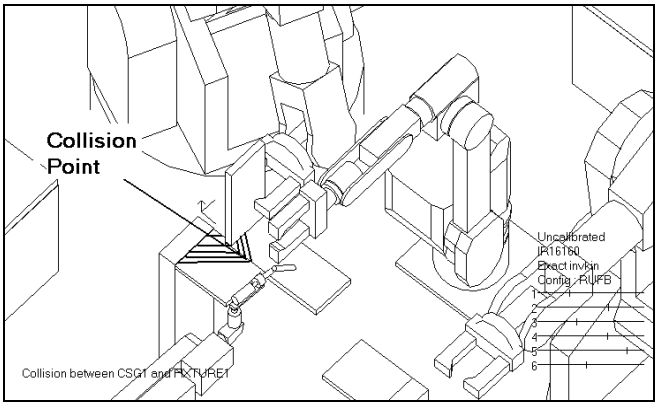

Figure.10: 3rd Collision Point in Case Study 2.

After this point, second level of optimization is initiated. The second state is reached the limits of the final state in 4 generations. The final placement errors are given in the following table.

Table.5: Positional Errors at the Final State.

Coordinate:	Error (mm):
X:	5
Y:	5
Z:	1

Totally 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 5 lines between the BEGIN and END commands as given below. The first 2 lines are from initial state to intermediate state, while the rest of them are for the recovery from intermediate state to final state. The change in the objective function is given in Table 6.

```
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
```

Table.6: 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

The change in the objective function from intermediate state to final state is smooth in this case. However, as it can be seen from Figure.11 the average fitness of the members dropped at the 8th generation since a new population is generated for the second level optimization process. It is realized that the average value increased between 8th and 10th generations and this helped the best fitness to increase in 11th generation.

It is noted that the second level of optimization converged in 4 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.

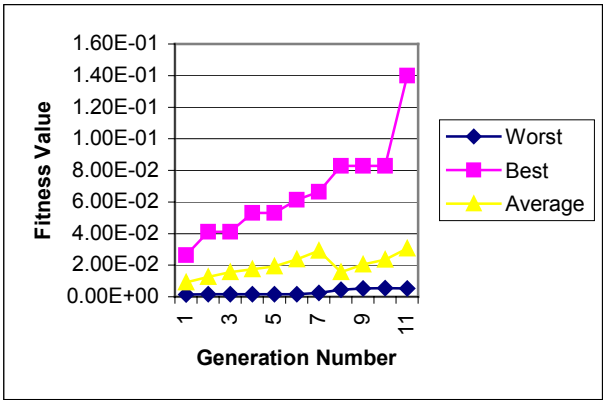

Figure.11: Optimization History of the 2nd Case Study.

DISCUSSIONS AND FUTURE WORK

In this paper an approach for the generation of robust recovery algorithms is presented. The implemented system is an extension on the infrastructure that was developed in our previous study. Since part misplacement errors are widely occurred during assembly process, recovery for the 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 procedure involves multi-level optimization process coupled with genetic programming. 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 less number of iterations than the single-step optimization case. Therefore the computation speed would be increased.

- Leads to a robust error recovery algorithm, which can be used for different error states (i.e. collision errors occurred at different points in an assembly line).

Although three states (n=3) are considered for each case study, this procedure can be applied to 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 in order to find robust recovery algorithms for each error case. The future studies are aimed on this type of error case analysis by using Monte-Carlo method.

REFERENCES

Abu-Hamdan M. G., El-Gizawy A. S., *"Computer Aided Monitoring System for Flexible Assembly Operations"*, Computers in Industry, Vol. 34, pp. 1-10, 1997.

Baydar C., Saitou K., *"Off-line Error Recovery Logic Synthesis in Automated Assembly Lines by using Genetic Programming,* 2000 Japan-USA Symposium on Flexible Automation, 2000a.

Baydar C., Saitou K., *"A Genetic Programming Framework for Error Recovery in Assembly Systems",* Genetic and Evolutionary Computation 2000 Conference, Las Vegas, Nevada, 2000b.

Brnyjolfsson S., Arnstrom A, *"Error Detection and Recovery in Flexible Assembly Systems"*, The International Journal of Advanced Manufacturing Technology, Vol.5, pp. 112-125, 1990.

Goldberg D., "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, Reading, MA, 1989.

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

Khodabandehloo K., *"Analyses of Robot Systems using Fault and Event Trees: Case Studies",* Reliability Engineering and System Safety 53, 247-264, 1996.

Hardy N., Barnes D., Lee M., *"Automatic Diagnosis of Task Faults in Flexible Manufacturing Systems"*, Robotica 7, 25-35, 1989.

Holland J., "Adaptation in Natural and Artificial Systems", MIT Press, Cambridge, MA, 1975.

Visinsky M.L., Cavallaro J.R., Walker I.D, *"Expert System Framework for Fault Detection and Fault Tolerance in Robotics"*, Computers in Electrical Engineering 20(5), 421-435, 1994.

Wirth R., Berthold B., Kramer A., Peter G., *"Knowledge Based Support of System Analysis for the Analysis of Failure Modes and Effects"*, Engineering Applications in Artificial Intelligence 9(3), 219-229, 1996.

Workspace v.4 Educational User Guide Manual, 1998.