ELSEVIER

Robotics and Autonomous Systems 13 (1994) 153–171

Robotics and Autonomous Systems

A dynamic recursive approach for autonomous inspection and reverse engineering

Tarek Sobh *, Jonathan Owen, Mohamed Dekhil

Computer Science Department, University of Utah, Salt Lake City, UT 84112, USA

Received 24 September 1993; revised 29 March 1994, 5 May 1994

Abstract

This paper addresses the application of discrete event dynamic systems (DEDS) for autonomous sensing and inspection as part of the reverse engineering process. A dynamic recursive context for DEDS is presented and its usage for managing a complex hybrid system which has continuous, discrete and symbolic aspects is illustrated. We suggest that the dynamic recursive context is aptly suited to controlling and observing the active inspection of machined parts using such a hybrid system.

Keywords: Autonomous systems; Discrete event systems; Inspection; Reverse engineering

1. Introduction

Reverse engineering is the process of constructing an accurate representation from sensed data. It can be represented by a closed loop system that consists of four main modules:

- Sensing,
- CAD Modelling,
- Manufacturing,
- Inspection.

This closed loop system is the framework we used to develop an integrated CAD/CAM/sensing system for inspection and reverse engineering. The process starts by constructing an initial CAD model using 2-D and 3-D vision, then the inspection module uses this model to drive a coordinate measuring machine (CMM). The results are used to increase the accuracy of the model. Additional sensing iterations could be made until the desired accuracy is obtained. Fig. 1 shows this closed loop system.

Most research in reverse engineering ([20,16,12,13,14,8,9]) concentrates on the sensing and fitting techniques required. Hsieh [15] describes a system which does sculptured surface reconstruction with a CMM. The focus of the work is on path planning and surface fitting. If errors occur while gathering data,

* Corresponding author. E-mail: sobh@wingate.cs.utah.edu

SSDI 0921-8890(94)00012-Q

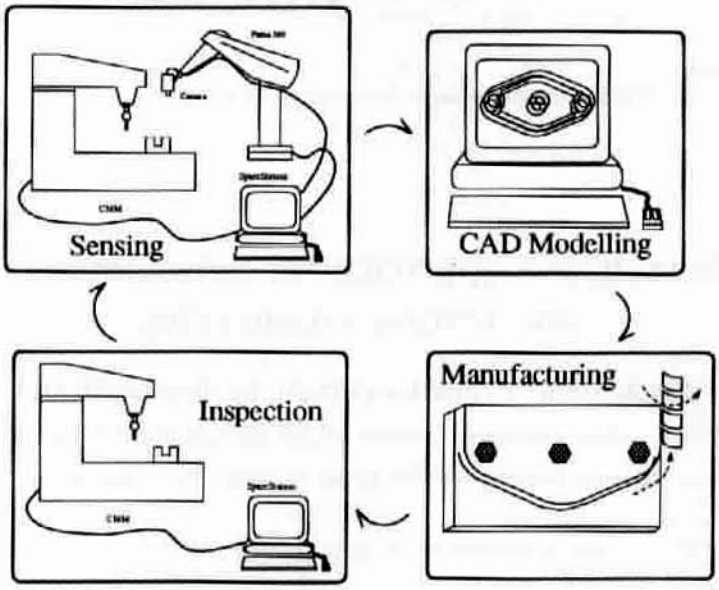


Fig. 1. Closed loop system for reverse engineering.

the system aborts and must be restarted. Van Thiel [30] describes an interactive CMM inspection system. The user is included as part of the control loop, and can abort inspections and call for explorations of particular features. This paper describes an approach that automatically gathers the sense data, processes it, and makes decisions based upon it for reverse engineering.

We use a recursive dynamic strategy for exploring machine parts. A discrete event dynamic system (DEDS) framework is designed for modeling and structuring the sensing and control problems. The dynamic recursive context for finite state machines (DRFSM) is a DEDS representation tailored to the recursive nature of the mechanical parts under consideration. For details about applying DEDS to various areas, see [3,18,10,2,22,21]. Some books on the subject include [11,7,1,4].

DRFSM is particularly useful for controlling the inspection module, and this has been an important aspect of our research.

2. Discrete event dynamic system control

Several representations can be used to model DEDS such as: finite automata, Petri nets, Markov chains, and queueing theory models. We chose finite state automata with partially observable events to model our system since it is suitable for the nature of this problem [17,5,19,24]. The discrete events that occur in the system trigger state transitions. Subsets of transitions can be disabled or enabled by the controller, depending on the application.

An example of a high-level DEDS controller for part inspection can be seen in Fig. 2. This finite state machine has some observable events that can be used to control the sequencing of the process. The machine remains in state A until a part is loaded. When the part is loaded, the machine transitions to state B where it remains until the part is inspected. If another part is available for inspection, the machine transitions to state A to load it. Otherwise, state C, the ending state, is reached. If an

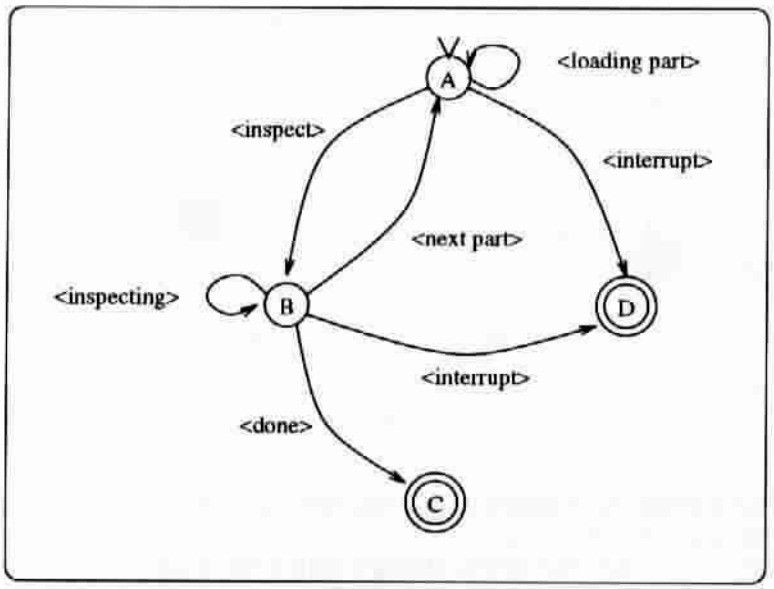


Fig. 2. A simple FSM.

interruption occurs, such as a misloaded part or inspection error, the machine goes to state D, the error state.

Our approach uses DEDS to drive a semi-autonomous visual sensing module that is capable of making decisions about the *state* of the inspection (e.g. the relation of the CMM probe to the part). This module provides both symbolic and parametric descriptions which can be used to interrupt the inspection or move to a new mode of inspection.

The applications of this work are numerous, including automatic inspection of mechanical or electronic components and reproduction of mechanical parts. The experience gained in applying DEDS to the inspection problem will allow us to study the subdivision of the solution into reliable, reversible, and an easy-to-modify software and hardware environments.

2.1. Modeling and constructing an observer

A DEDS framework is used to model the tasks that the autonomous observer system executes. This model is used as a high level structuring technique to preserve and make use of the information we know about the way in which a mechanical part should be explored. The state and event description is associated with different visual cues; for example, appearance of objects, specific 3-D movements and structures, interaction between the touching probe and part, and occlusions. A DEDS observer serves as an intelligent sensing module that utilizes existing information about the tasks and the environment to make informed tracking and correction movements and autonomous decisions regarding the state of the system.

To be able to determine the current state of the system we need to observe the sequence of events occurring in the system and make decisions regarding the state of the automaton. State ambiguities are allowed to occur, however, they are required to be resolvable after a bounded interval of events. In a *strongly output stabilizable* system, the state of the system is known at bounded intervals and allowable

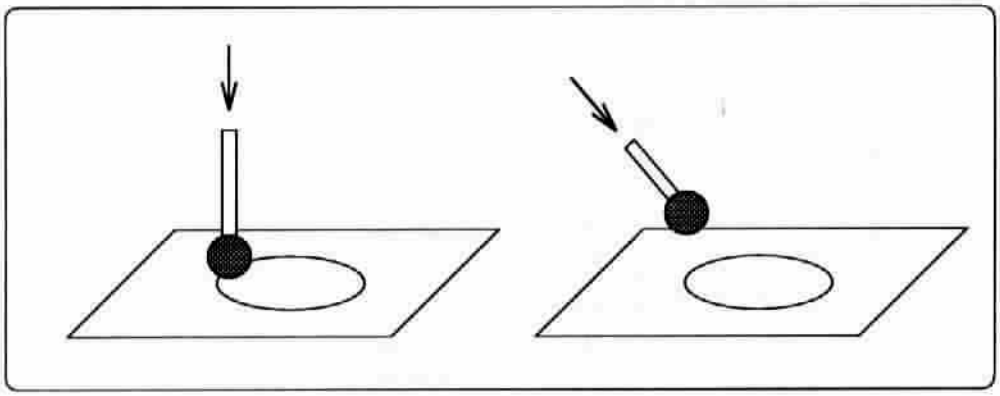


Fig. 3. Bad approach vector.

events can be controlled (enabled or disabled) in a way that ensures return in a bounded interval to one of a desired and known set of states.

One of the objectives is to make the system strongly output stabilizable and/or construct an observer to satisfy specific task-oriented visual requirements. Many 2-D visual cues for estimating 3-D world behavior can be used. Examples include: image motion, shadows, color and boundary information. The uncertainty in the sensor acquisition procedure and in the image processing mechanisms should be taken into consideration to compute the world uncertainty.

2.2. Error states and sequences

The observer framework can be utilized for recognizing error states and sequences. This recognition task will be used to report on *visually incorrect* sequences. In particular, if there is a pre-determined observer model of a particular inspection task under observation, then it would be useful to determine if something goes wrong with the exploration actions. The goal of this reporting procedure is to alert the operator or autonomously supply feedback to the inspecting robot so that it can correct its actions.

Some examples of errors that might occur while inspecting based on a reverse engineered model include:

- occlusions between the observer camera and the part or probe;
- inappropriate approach vector position or orientation (see Fig. 3);

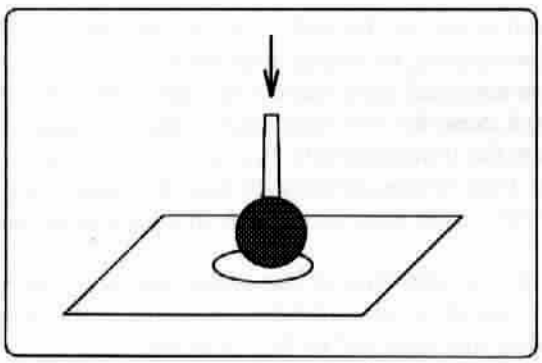


Fig. 4. Probe diameter too large.

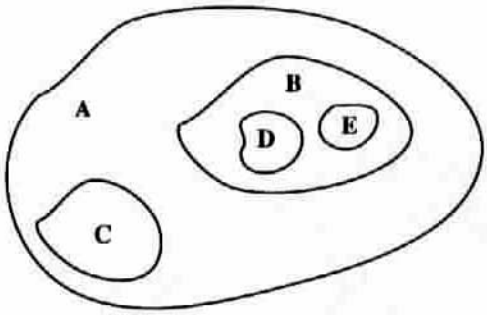


Fig. 5. A hierarchy example.

- inappropriate probe size (see Fig. 4);
- motion too rapid;
- motion too slow ("frozen" or "timeout").

The correct sequences of automata state transitions can be formulated as the set of strings that are *acceptable* by the observer automaton. This set of strings represents precisely the language describing all possible visual task evolution steps.

3. The dynamic recursive context for finite state machines

The dynamic recursive context for finite state machines (DRFSM) is a form of DEDS which is specifically adapted to representing multi-level recursive processes. Multi-level processes are any tasks which are done repetitively with different parameters.

DRFSM can be used to exploit the recursive nature of many machined parts. Many machined features have similar inspection strategies. By using the same strategy for different features within a complicated part, we can reduce the number of control states needed to inspect it to a manageable amount.

3.1. Recursive representation for machine parts

We propose a representation for machine parts which is composed of open and closed contours and the relationships between them. For example, the relations between the enclosed areas of the contours shown in Fig. 5 would be expressed as follows:

$B \subset A$,
$C \subset A$,
$D \subset B$,
$D \subset A$,
$E \subset B$,
$E \subset A$.

These relations can be represented by a graph, see Fig. 6. Using a depth-first search, a string representation can be obtained from this graph. This string can be represented by a tree as shown in Fig. 7. This string will guide the DRFSM to explore this part recursively. The string representation for the above example will be:

$$A(B(D(), E()), C()).$$

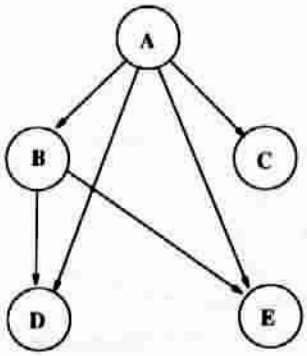


Fig. 6. The graph associated with the example.

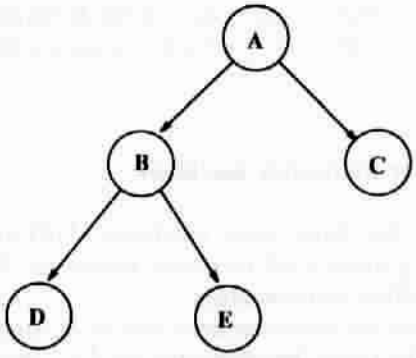


Fig. 7. The tree associated with the example.

The algorithm used to determine these relationships was based on region growing. To determine if a feature, F_1, lies within another, F_2, the outlines of both features are copied into a blank image. If F_2 overwrites F_1 (as was the convention in our system), border intersections will be considered inside. [1]

Each pixel in the outline of F_1 is then "grown" in 4 or 8 directions, marking pixels as visited, stopping when one of the following is encountered:

- F_2's outline,
- a previously visited pixel,
- the image border.

If the image border is not encountered, F_1 is within F_2.

This method depends upon F_2 being a closed curve, with the boundary 4- or 8-connected (depending upon the growing algorithm). Additionally, F_2 must be at least one pixel away from the image border. Advantages of this algorithm are that it is not necessary to determine if any particular pixel is inside or outside of F_2 and that F_1 need not be closed.

[1] If F_1 overwrites F_2 (an equally valid convention), border intersections will cause F_2 to become an open curve, ensuring that $F_1 \neg \subset F_2$.

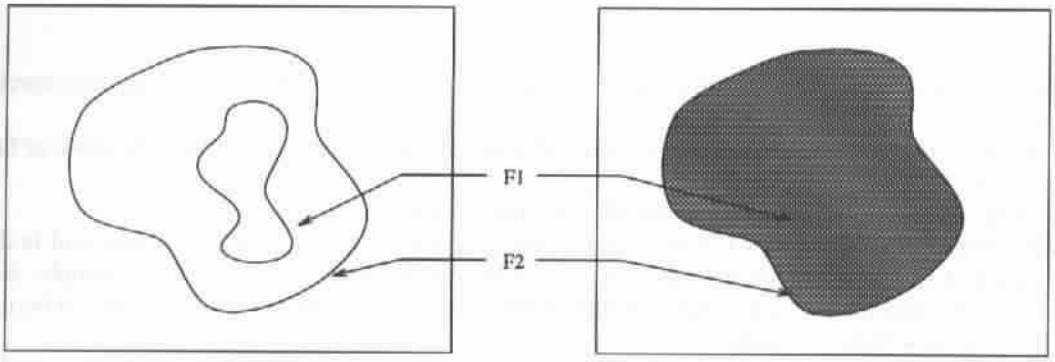


Fig. 8. F_1 inside F_2.

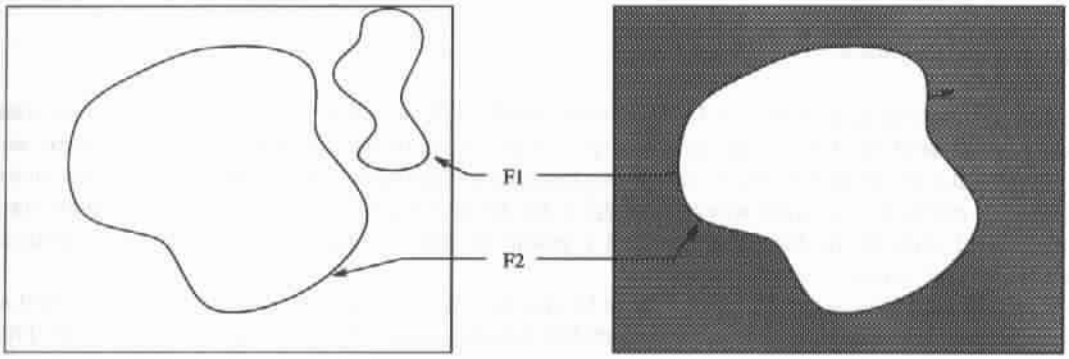


Fig. 9. F_1 outside F_2.

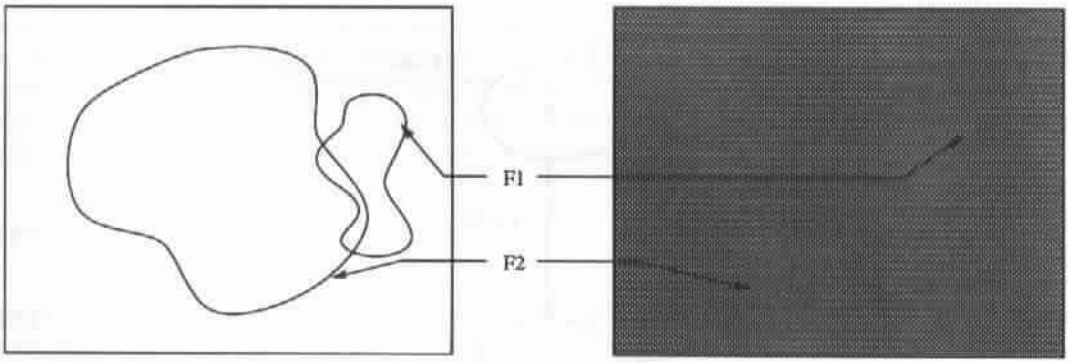


Fig. 10. Intersection, F_1 not inside F_2.

The right side of Fig. 8 shows the result of applying this algorithm to test if a curve, F_1, is inside another curve, F_2. The growth of F_1 is completely blocked from reaching the image boundary by F_2. In Fig. 9, F_1 grows to the border and is thus considered to be outside. Fig. 10 shows the case when F_1 and F_2 intersect. F_1's growth fills the entire image, both inside and outside of F_2.

3.2. Definitions

- *Variable transition value:* Any variable value that depends on the level of recursion. In our experiments, we use two:
  - *V1* is the distance threshold that determines whether the probe is considered to be close or far from a feature,
  - *V2* is the allowable length of time for the machine to wait for an event to occur.
- *Variable transition vector* (VTV): The vector containing all variable transition values, and is dynamically changed in the event of a transition to a different level of recursion. For example, to begin exploring a feature that is "contained" within another, the each value in the VTV should be updated to reflect the new feature's scale.
- *Dead-end state:* A state that does not call any other state (no transition arrows come out of it). In DRFSM, when this state is reached, it means to go back to a previous level, or quit if it is the first level. This state is usually called the Error-trapping state. It is desirable to have several dead-end states to represent different types of errors that can happen in the system.

3.3. DRFSM representation

The same notation and terms of the ordinary FSMs will be used, but we will introduce some new notation to represent recursive and variable transitions. First, there is a new type of transition, as shown in Fig. 11; (from state B to A), this is called the Recursive Transition (RT). This figure shows an example where state A might be the state where a probe is far from a feature, state B where it is near; state C an error state; and state D the final state. d and t would be the distance from the current feature and the time since the last event.

A recursive transition arrow from one state to another means that the transition from the first state to the second state is done by a recursive call to the second state after changing the Variable Transition

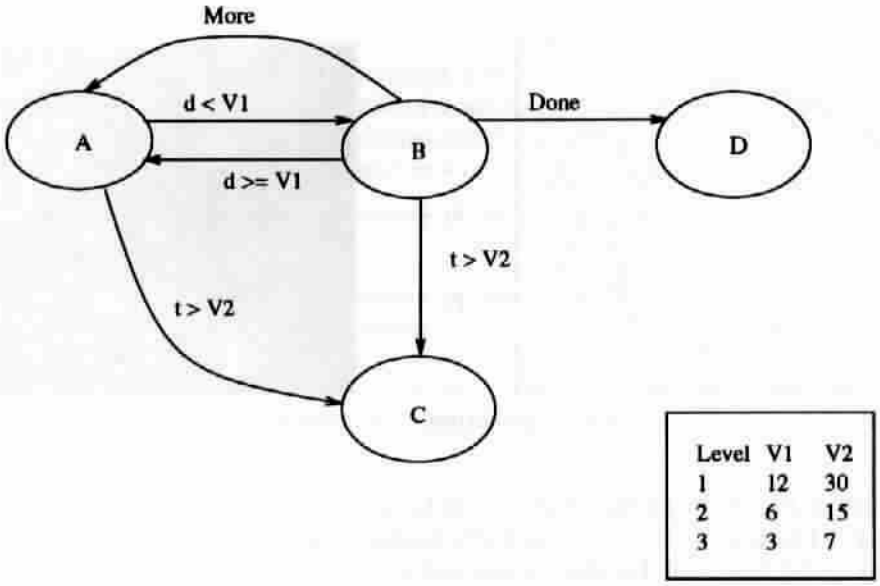


Fig. 11. A simple DRFSM.

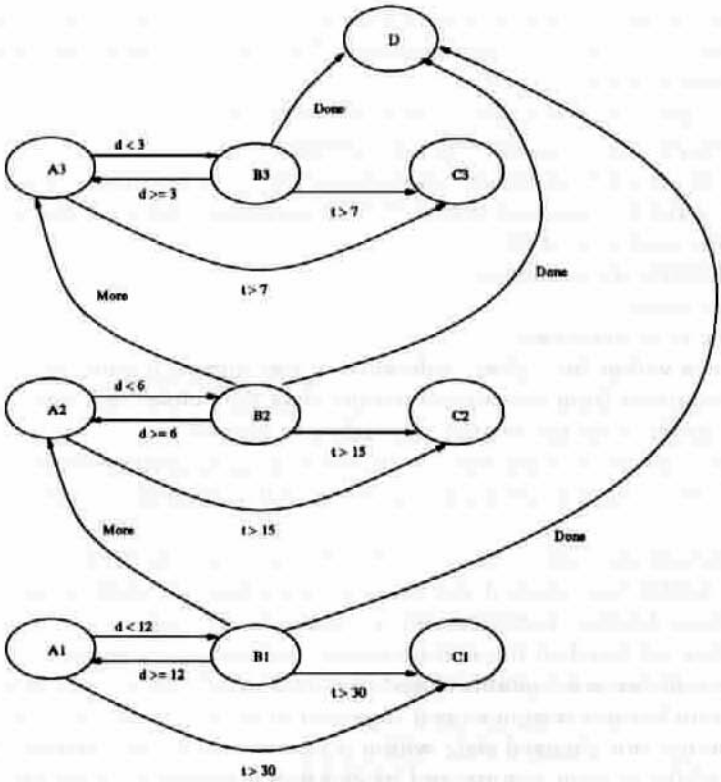


Fig. 12. Flat representation of a simple DRFSM.

Vector. Second, the transition condition from a state to another may contain variable parameters according to the current level, these variable parameters are distinguished from the constant parameters by the notation V_n. All variable parameters of all state transitions constitute the Variable Transition Vector. It should be noticed that nondeterminism is not allowed, in the sense that it is impossible for two concurrent transitions to occur from the same state. Fig. 12 is the equivalent FSM representation (or the flat representation) of the DRFSM shown in Fig. 11, for three levels, and it illustrates the compactness and efficiency of the new notation for this type of process.

3.4. A graphical interface for developing DRFSMs

A graphical interface was developed to allow quick and easy means for modifying the DRFSM which drives the inspection process. This was accomplished by modifying an existing reactive behavior design tool, GIJoe, to accommodate producing the code of DRFSM DEDS.

GIJoe was designed by Mark Bradakis at the University of Utah [6]. It allows the user to graphically draw finite state machines, and output them as C code. The graphical user interface allows the user to place states and transitions with a mouse. Transitions can be labelled with boolean combinations of symbols, such as "A and B or C". When the state machine is complete, the user selects a start state and clicks a "Compile" button to output C code which duplicates the structure of the machine. The machine can be saved and later modified for different applications.

The code output by the original GIJoe has an iterative structure that is not conducive to the recursive formulation of dynamic recursive finite state machines. Therefore, it was decided to modify GIJoe to suit our needs. Modifications to GIJoe include:

- Output of recursive rather than iterative code to allow recursive transitions.
- Modificaton of string parsing to accept recursive transition specification.
- Encoding of an event parser to prioritize incoming events from multiple sources.
- Implementation of variable transition vector (VTV) acquisition (when making recursive transitions.) The VTV is currently read from a file.

Currently acceptable events are as follows:

- Probe – probe is the scene.
- NoProbe – no probe is in the scene.
- ProbeClose – probe is within the "close" tolerance to the current feature specified by the VTV.
- ProbeFar – probe is farther from the current feature than the "close" tolerance specified by the VTV.
- ProbeOnFeature – probe is on the feature (according to vision).
- ProbeNotOnFeature – probe is close, but not on the feature (according to the vision).
- VisionProblem – part string has changed, signifying that a feature is occluded (need to move the camera).
- ProblemSolved – moving the camera has corrected the occlusion problem.
- TouchedFeature – probe has touched the feature (according to touch sensor,) feature is explored quantitatively and exact feature measurements are taken in this state.
- NoTouch – probe has not touched the feature (according to touch sensor).
- ClosedRegion – current feature contains closed region(s) to be inspected (recursively),
- OpenRegion – current feature contains open region(s) to be inspected (iteratively),
- TimeOut – machine has not changed state within a period of time specified by the VTV.
- Done – inspection of the current feature and its children is complete, return to previous level.

Additional events require the addition of suitable event handlers. New states and transitions may be added completely within the GIJoe interface. The new code is output from GIJoe and may be linked to the inspection utilities with no modifications.

The following is a transcript showing the exploration of two closed regions A and B, with A containing B:

```
inspect[5] ~/DEDS⇒ bin/test_drfsm
  enter the string:  A(B())
A(B())

THE VARIABLE TRANSITION VECTOR

  100.000000  50.000000
in state A
 has the probe appeared? n                    % NoProbe is true
 has the probe appeared? n
 has the probe appeared? y                    % Probe is true
in state B
 has the probe appeared? y
 enter the distance from probe to A: 85  % Probe is Far
 has the probe appeared? y
 enter the distance from probe to A: 45  % Probe is Close
```

```
in state C
 enter the string: A(B())
 enter the distance from probe to A: 10
 is the probe on A? y
in state D
 is the probe on A? y                    % Probe on Feature
                                         % (Measure Feature Parameters)
 has touch occurred? y
in state E
Making recursive call...

THE VARIABLE TRANSITION VECTOR

  100.000000  50.000000
in state A
 has the probe appeared? y               % Probe is true
in state B
 has the probe appeared? y
 enter the distance from probe B: 95     % Probe is Far
 has the probe appeared? y
 enter the distance from probe to B: 45  % Probe is Close
in state C
 enter the stirring: A(B())
 enter the distance from probe to B: 10
 is the probe on B? y
in state D
 is the probe on B? y                    % Probe on Feature
                                         % (Measure Feature Parameters)
 has touch occurred? y
in state E
in state END
in state END

Inspection Complete.

inspect[6] ~/DEDS⇒
```

The obtained results when linked with the rest of the experimental code were as expected. Future modifications may include the addition of "output" on transitions, such as "TouchOccurred/UpdateModel", allowing easy specification of communication between modules. It should be clear, however, that the code generated by GIJoe is only a skeleton for the machine, and has to be filled by the users according to the tasks assigned to each state.

In general, GIJoe proved to be a very efficient and handy tool for generating and modifying such machines. By automating code generation, one can reconfigure the whole inspection process without being familiar with the underlying code (given that all required user-defined events and modules are available).

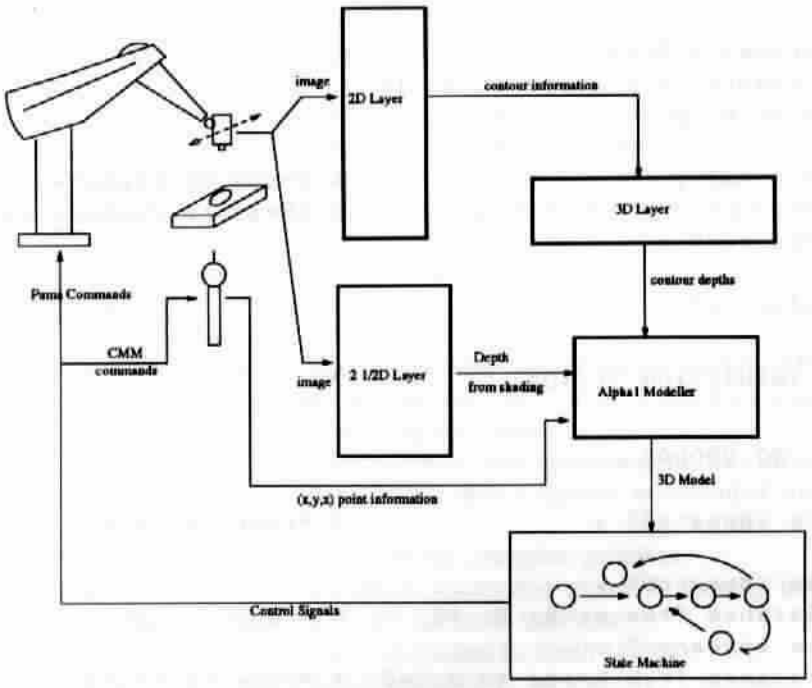


Fig. 13. Inspection system overview.

4. Experiments

In conducting our experiments, we use a B/W CCD camera mounted on a Puma 560 robot arm, that observe and guide the interaction between the CMM probe and the machined part (see Fig. 13.) In order for the state machine to provide control, it must be aware of state changes in the system. As inspection takes place, the camera supplies images that are interpreted by a set of 2-D and 3-D vision processing algorithms and used to drive the DRFSM. These algorithms are described in greater detail in other publications [29,23,26,28,25,27], but include thresholding, edge detection, region growing, stereo vision, etc. The robot arm is used to position the camera in the workplace and move in the case of occlusion problems.

An early experiment was run without using the robot to move the camera, and with a "hand-generated" automaton. Our latest experiment uses the robot and GIJoe-generated automata.

The object of these experiments was to test the operation of the visual system with the state machine. Two facets of this were the generation of an initial model from stereo vision and the generation of events that describe a probe's relationship to features in that model.

This stereo process used the Puma arm to gather pairs of images. The resulting model was used to determine feature relationships used in the DEDS controller. The models shown are from this initial visual inspection.

The event generation method, consisting of 2-D image processing routines, was used to detect the relationship of a simulated (handheld) CMM probe to the features in the initial model. These events were processed by the controller, which output text messages guiding the experimenter to move the probe or indicate that a touch had occurred.The DRFSM generated by GIJoe is shown in Fig. 14. This machine has the following states:

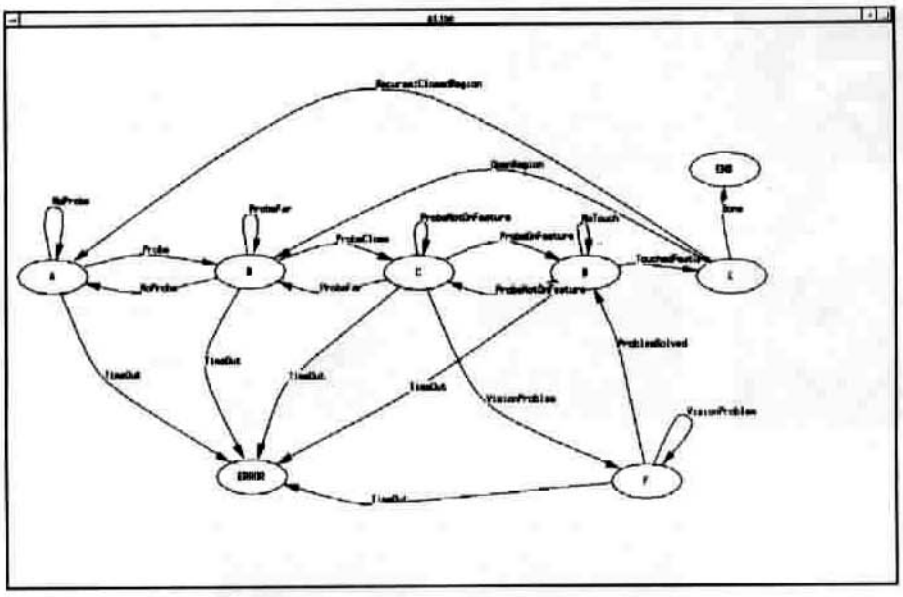


Fig. 14. GIJoe window w/DRFSM.

- *A*: The initial state, waiting for the probe to appear.
- *B*: The probe appears, and waiting for it to be close. Here, "close" is a measure of the distance between the probe and the current feature, since it depends on the level of the recursive structure. For example, the distance at the first level, which represents the outer contours or features, is larger than that of the lower levels. The closeness tolerance is taken from the VTV.
- *C*: Probe is close, but not on feature.
- *D*: The probe appears to be on feature in the image, and waiting for physical touch indicated from the CMM machine.

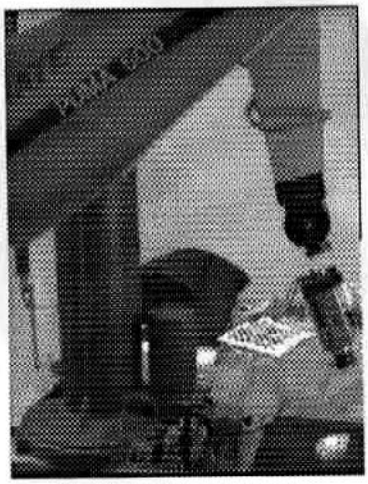


Fig. 15. Experimental set-up.

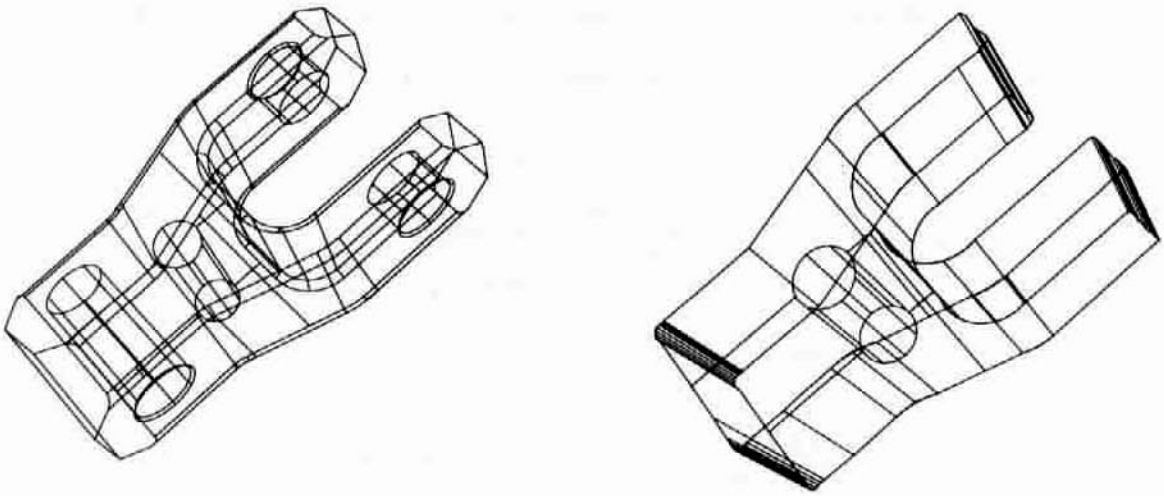


Fig. 16. Original and reverse-engineered part models.

- *E*: Physical touch has happened (and the CMM measurements for the feature parameters are recorded and saved for updating the CAD model). If the current feature represents a closed region, the machine goes one level deeper to get the inner features by a recursive call to the initial state after changing the variable transition parameters. If the current feature was an open region, then the machine finds any other features in the same level.
- *F*: This state is to solve any vision problem happens during the experiment. For example, if the probe is occluding one of the features, then the camera position can be changed to solve this problem.

Fig. 17. Original and reproduction.

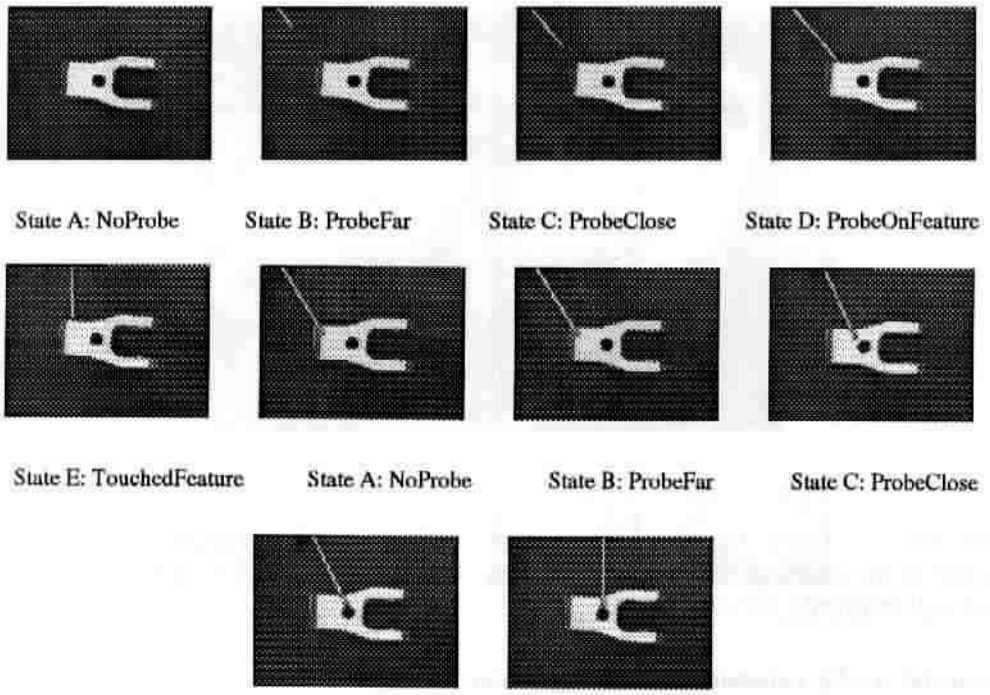


Fig. 18. Bracket sequence.

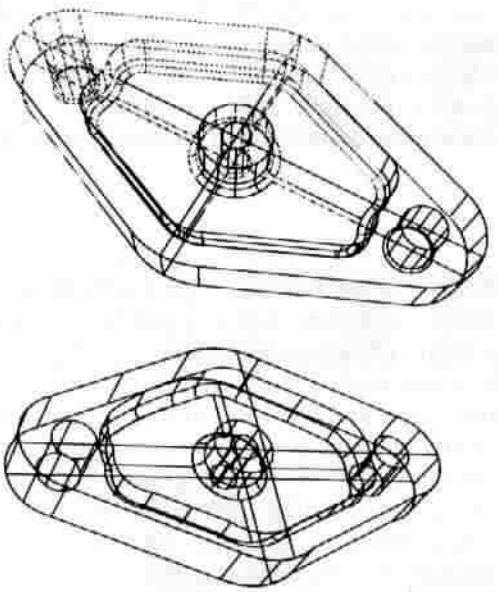


Fig. 19. Original and vision-reverse engineered models.

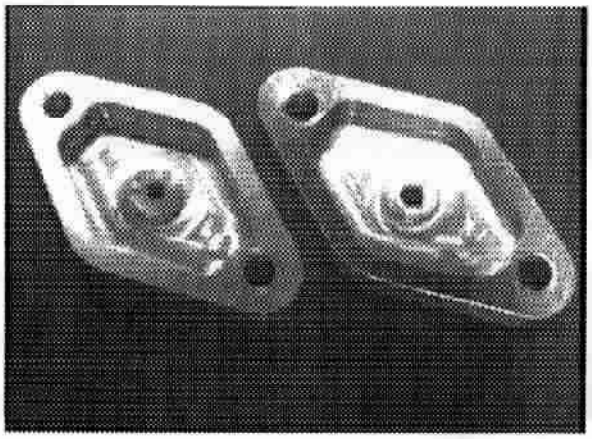


Fig. 20. Original and vision-reverse engineered parts.

- ERROR: There is a time limit for each part of this experiment, specified in the VTV. If, for any reason, one of the modules does not finish in time, the machine will go to this state, which will report the error and terminate the experiment.

4.1. Experimental results, automated bracket inspection

A metal bracket was used in the experiment to test the inspection automaton. The piece was placed on the inspection table within view of the camera (see Fig. 15).

The machine was brought on line and execution begun in State A, the start state. After initiating the inspection process, the DRFSM transitioned through states until the probe reached the bracket boundary. The state machine then called for the closed region to be recursively inspected until finally, the hole was explored and the machine exited cleanly. The sequence is shown in Fig. 18.

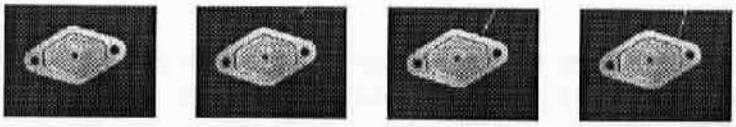
The original part and the resulting reverse-engineered part are shown in Figs. 16 (wireframes) and 17 (rendered images). Notice that the two side holes and a portion of the bracket were not sensed correctly, as a simple strategy was used to sense from only one direction. In the next experiment, a more complicated model is sensed with a more sophisticated sensing and modelling strategy.

4.2. Experimental results, cover plate

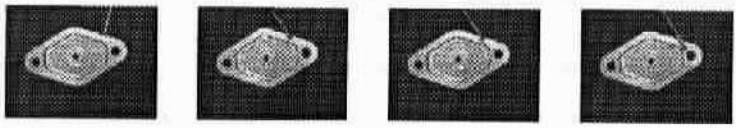
A second experiment was run in a similar fashion, using a part similar to the fuel pump cover from a Chevrolet engine. This piece offers interesting features and has a complex recursive structure which allowed us to test the recursive nature of the state machine.

The sensing strategy used here was more robust than in the previous experiment. Detected feature contours were sensed with stereo vision and used to build up a feature-based α_1 model. This model was then used to semi-automatically machine a reproduction of the part. The original and reverse-engineered wireframe models are shown in Fig. 19. A photograph of the original and reproduction is shown in Fig. 20. For more detail on the sensing strategy, please see [29].

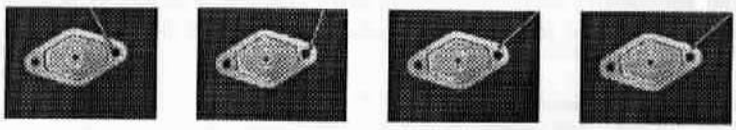
The inspection sequence corresponding to this experiment is shown in Fig. 21. Shown there, the DRFSM transitions correctly through the inspection of the outside profile (depth of recursion = 0), a hole (1), a profile pocket (1), a hole (2), and another hole (1).



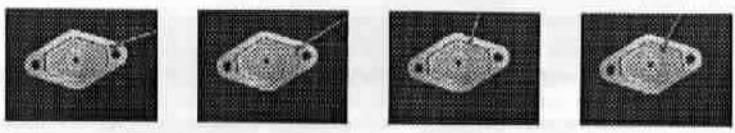
State A: NoProbe State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature



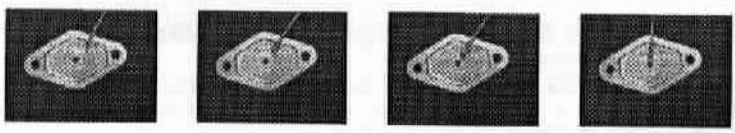
State E: TouchedFeature State A: NoProbe State B: ProbeFar State C: ProbeClose



State D: ProbeOnFeature State E: TouchedFeature State A: NoProbe State B: ProbeFar



State C: ProbeClose State D: ProbeOnFeature State E: TouchedFeature State A: NoProbe



State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature State E: TouchedFeature

State A: NoProbe State B: ProbeFar State C: ProbeClose State D: ProbeOnFeature

State E: TouchedFeature

Fig. 21. Cover sequence.

5. Conclusions

We have constructed a framework for autonomous inspection and reverse engineering which utilizes discrete event dynamic system control. We used a dynamic recursive context for DEDS (DRFSM) that was suited to the recursive nature of the problem area, parts composed of machined features.

Several experiments were performed to illustrate the utility of this framework, and we believe that DRFSM provide robust control in the domain of autonomous sensing and inspection for machine parts. An interactive package has been developed which allows a user to graphically generate DRFSM automata.

Acknowledgments

This work was supported by the Advanced Research Projects agency under Army Research Office grant number DAAH04-93-G-0420. Also, DARPA grant N00014-91-J-4123, NSF grant CDA 9024721, and a University of Utah Research Committee grant. All opinions, findings, conclusions or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] F. Bacelli, G. Cohen, G. Olsder and J. Quadrat, *Algebraic and Stochastic Analysis of Timed Discrete Event Systems* (Wiley, New York, 1991.

[2] R. Bajcsy and T. Sobh, A framework for observing a manipulation process, Tech. Rep. MS-CIS-90-34, GRASP Lab 216, Department of Computer and Information Science, University of Pennsylvania, June 1990.

[3] S. Balemi, Control of discrete event systems: Theory and application, Ph.D. thesis, Swiss Federal Institute of Technology, May 1992.

[4] J. Banks and J. Carson, *Discrete-Event System Simulation* (Prentice-Hall, Englewood Cliffs, NJ, 1984).

[5] A. Benveniste and P.L. Guernic, Hybrid dynamical systems theory and the signal language, *IEEE Transactions on Automatic Control* 35 (5) (1990).

[6] M.J. Bradakis, Reactive behavior design tool, Master's thesis, Computer Science Department, University of Utah, January 1992.

[7] C. Cassandras, *Discrete Event Systems: Modeling and Performance Analysis*, Aksen Associated Series in Electrical and Computer Engineering (IRWIN, Homewood, IL, 1993).

[8] Y. Chen and G. Medioni, Object modelling by registration of multiple range images, *International Journal of Image and Vision Computing* 10 (3) (1992) 145–155.

[9] Y. Chen and G. Medioni, Integrating multiple range images using triangulation, *Image Understanding Workshop* (April 1993). (Defense Advanced Research Projects Agency, Software and Intelligent Systems Office) 951–958.

[10] Y. Ho, Discrete event dynamical system and its application to manufacturing, *Distributed Computer Control Systems: Proc. Third IFAC Workshop*, Beijing, China (Aug. 1981).

[11] Y. Ho and X. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems* (Kluwer Academic Publishers, Norwell, MA, 1991).

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Surface reconstruction from unorganized points, *Computer Graphics, SIGGRAPH '92*, (July 1992), Vol. 26.

[13] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Mesh optimization, *Computer Graphics, SIGGRAPH '93* (Aug. 1993), Vol. 27.

[14] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald and W. Stuetzle, Piecewise smooth surface reconstruction, *Computer Graphics, SIGGRAPH '94* (1994).

[15] Y.C. Hsieh, Reconstruction of sculptured surfaces using coordinate measuring machines, Master's thesis, Mechanical Engineering Department, University of Utah, June 1993.

[16] S. Motavalli and B. Bidanda, A part image reconstruction system for reverse engineering of design modifications, *J. Manufacturing Systems* 10 (5) (1991) 383–395.

[17] A. Nerode and J.B. Remmel, A model for hybrid systems, *Proc. Hybrid Systems Workshop*, Mathematical Sciences Institute (May 1991), Cornell University.

[18] G. Olsder, Applications of the theory of discrete event systems to array processors and scheduling in public transportation, *28th IEEE CDC* (Dec. 1989).
[19] C.M. Özveren, Analysis and control of discrete event dynamic systems: A state space approach, Ph.D. thesis, Massachusetts Institute of Technology, Aug. 1989.
[20] B. Sarkar and C. Menq, Smooth-surface approximation and reverse engineering, *Computer Aided Design* 23 (9) (1991) 623–628.
[21] T. Sobh, Active observer: A discrete event dynamic system model for controlling an observer under uncertainty, Ph.D. thesis, University of Pennsylvania, Dec. 1991.
[22] T. Sobh, R. Bajcsy and J. James, Visual observation for hybrid intelligent control implementation, *31st IEEE CDC*, Tuscon, AZ (Dec. 1992).
[23] T. Sobh, C. Jaynes, M. Dekhil and T. Henderson, *Intelligent Systems: Safety, Reliability, and Maintainability Issues*, ch. Automated inspection and reverse engineering (Springer-Verlag, Berlin, 1993) 95–122.
[24] T.M. Sobh and R. Bajcsy, A model for observing a moving agent, *Proc. Fourth International Workshop on Intelligent Robots and Systems (IROS '91)*, Osaka, Japan (Nov. 1991).
[25] T.M. Sobh, M. Dekhil, C. Jaynes and T. Henderson, A perception framework for inspection and reverse engineering, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR '93)*, New York (June 1993).
[26] T.M. Sobh, M. Dekhil and J.C. Owen, Discrete event control for inspection and reverse engineering, *IEEE International Conference on Robotics and Automation*, San Diego, CA (May 1994).
[27] T.M. Sobh, C. Jaynes and T. Henderson, A discrete event framework for intelligent inspection, *IEEE International Conference on Robotics and Automation*, Atlanta (May 1993).
[28] T.M. Sobh, J.C. Owen, M. Dekhil, C. Jaynes and T. Henderson, Industrial inspection and reverse engineering, *IEEE 2nd CAD-Based Vision Workshop*, Pittsburgh (February 1994).
[29] T.M. Sobh, J.C. Owen, C. Jaynes, M. Dekhil and T.C. Henderson, Active inspection and reverse engineering, Tech. Rep. UUCS-93-007, Department of Computer Science, University of Utah, March 1993.
[30] M. Van Thiel, Feature based automated part inspection, Master's thesis, University of Utah, 1993.

Tarek M. Sobh received the B.Sc. in Engineering degree with honors in Computer Science and Automatic Control from the Faculty of Engineering, Alexandria University, Egypt in 1988, and M.S. and Ph.D. degrees in Computer and Information Science from the School of Engineering, University of Pennsylvania in 1989 and 1991, respectively. He is currently a Research Assistant Professor of Computer Science at the Department of Computer Science, University of Utah, and the Co-Chairman of the Discrete Event Dynamic Systems (DEDS) Technical Committee of the IEEE Robotics and Automation Society. Dr. Sobh was a Research Fellow at the General Robotics and Active Sensory Perception (GRASP) Laboratory of the University of Pennsylvania during 1989–1991. His background is in the fields of computer science and engineering, control theory, robotics, computer vision and signal processing. His current research interests include reverse engineering and industrial inspection, CAD/CAM and active perception under uncertainty, robots and electrochemical systems prototyping, and automation for genetics applications. He has published over 50 journal and conference papers, book chapters, and technical reports in those areas. Professor Sobh is also interested in developing theoretical and experimental tools to aid performing adaptive goal-directed robotic sensing for modeling, observing and controlling interactive agents in unstructured environments. Dr. Sobh is a Licensed Professional Electrical Engineer in the State of Utah, a member of ACM, IEEE, Tau Beta Pi, Sigma Xi, and Phi Beta Delta honor societies.

Jonathan Owen received a B.S. in Mechanical Engineering in 1986 from Rice University. He was employed by the McDonnell-Douglas Corporation as an engineer in its Structural Research Department in St. Louis until 1989. Some of the projects he worked on included development and support of a pre- and post-processor for finite element analysis and heat exchanger design for the National Aerospace Plane (NASP). He then worked with Noetic Technologies and later the MacNeal-Schwendler Corporation as a software engineer, developing *P*-version finite element interfaces. In 1991, Jonathan started pursuing an M.S. in computer science at the University of Utah. He has since concentrated on computer vision and is currently working on his thesis, "Feature-based Reverse Engineering".

Mohamed Dekhil received the B.Sc. in Engineering degree with honors in Computer Science and Automatic Control from Alexandria University, Egypt in 1988, and an M.S. degree in Computer Science from the School of Engineering, University of Utah in 1993. He is currently a Ph.D. student at the Department of Computer Science, University of Utah. Mr. Dekhil was a Research Fellow at the National Research Center, Cairo, Egypt until 1992, and he also worked as a software engineer at Systems Research Egypt (SRE), a major software house in Cairo. Mr. Dekhil is interested in robot design and simulation, prototyping environment for robotic manipulators, inspection and reverse engineering, and multisensor integration.