Discrete Event Control for Inspection and Reverse Engineering

Tarek M. Sobh, Mohamed Dekhil, and Jonathan C. Owen [1]

UUCS-94-005

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

February 17, 1994

Abstract

We address the problem of intelligent sensing in this work. In particular, we use discrete event dynamic systems (DEDS) to guide the sensing of mechanical parts for industrial inspection and reverse engineering.

[1]This work was supported by ARPA under ARO grant number DAAH04-93-G-0420, 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. This report appears as a paper in the proceedings of the 1994 IEEE International Conference on Robotics and Automation.

Discrete Event Control for Inspection and Reverse Engineering

Tarek M. Sobh, Mohamed Dekhil, and Jonathan C. Owen *

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

Abstract

We address the problem of intelligent sensing in this work. In particular, we use discrete event dynamic systems (DEDS) to guide the sensing of mechanical parts for industrial inspection and reverse engineering.

1 Introduction

Reverse engineering is essentially the problem of constructing a model from sensed information. To do so within the tolerances needed in most manufacturing applications requires sophisticated sensing such as with a coordinate measuring machine (CMM). A CMM uses a robot arm to move a relatively delicate sensor in contact with the object to be sensed. To navigate this sensor efficiently and without collision requires some information about the object to be sensed. In an inspection situation, this information is typically the CAD model from which the part was manufactured. In a reverse engineering situation, the CAD model is not available, and this information must come from some other form of non-contact sensing such as intensity or range sensors. This information will typically be less accurate than the original CAD model unless considerable time and expense is spent on non-contact sensing. A robust control system can be used to make up for this.

Unifying the control of hybrid systems can pose a difficult problem. We feel that discrete event dynamic systems (DEDS) are very appropriate for the control of such systems. We have implemented such a strategy and introduce the dynamic recursive context for finite state machines (DRFSM) as a new DEDS tool for utilizing the recursive nature of the mechanical parts under consideration.

*This work was supported by ARPA under ARO grant number DAAH04-93-G-0420, 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.

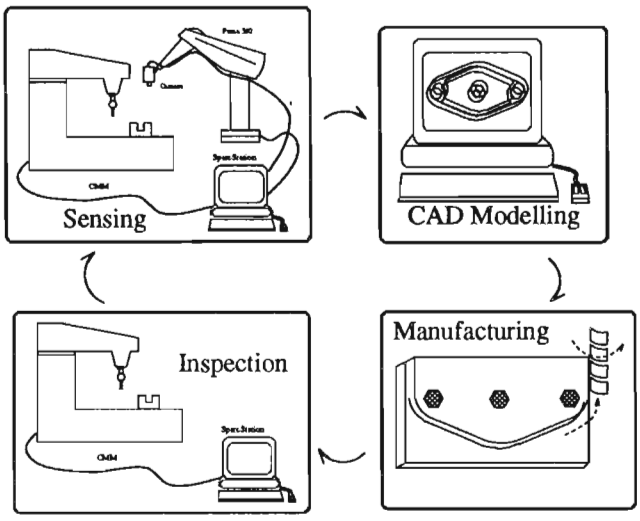


Figure 1: A Closed Loop System

2 Reverse Engineering and Inspection System

An integrated CAD/CAM/sensing system can be described as a closed loop (see Figure 1). We have developed a sensing system for reverse engineering which uses 2-d and 3-d vision to construct a CAD model of a part[4]. This sensing system interfaces with the University of Utah's α_1 modelling system which has a semi-automatic interface to automated manufacturing equipment.

Two-dimensional image processing routines are used to segment an image which contains an industrial part, and find closed contours in a pair of images (motion provided by robot arm). A stereo vision algorithm is used to obtain three dimensional data on the contours, and fitting and constraints are used to build a CAD model from the data. One of the resulting reverse-engineered parts is shown in Figure 2. The models used to manufacture these parts are shown in Figure 3. Note that the original model was not used in deriving the reverse engineered model, only sense data from the part itself.

Although the part is quite similar to the original, we would like to "close the loop" and use automated inspection with a CMM for a more accurate representa-

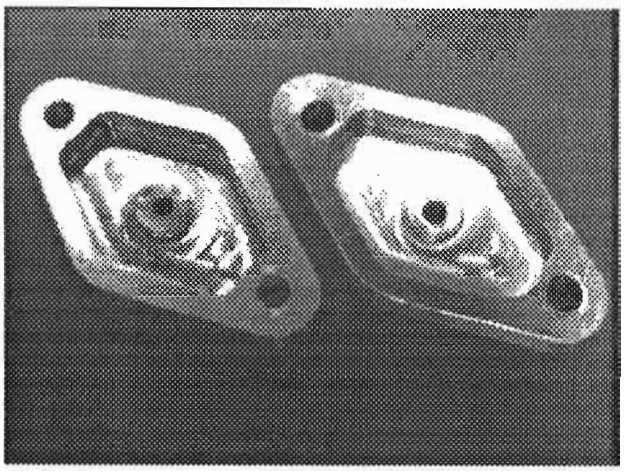


Figure 2: Original and Vision-Reverse Eng'd Parts

tion. To do this, we will inspect the original part, using our vision-derived model as the baseline. Doing so will require a robust control system as is described in the following sections.

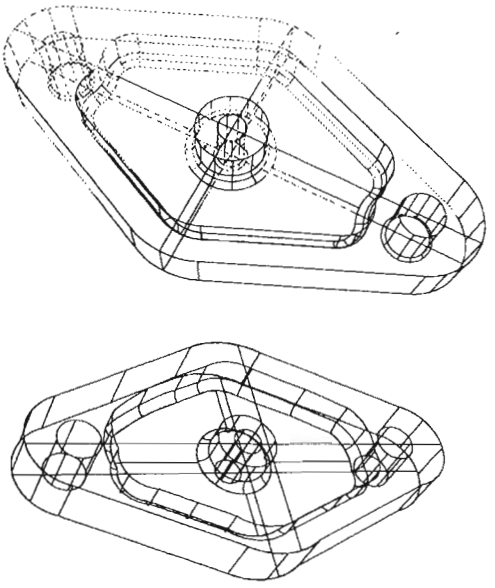


Figure 3: Original and Vision-Reverse Eng'd Models

3 DEDS Control for Inspection

DEDS are dynamic systems in which discrete events occur. If modeled by state machines (see Figure 4), these discrete events would trigger state transitions. These systems are typically asynchronous, and can be used as control models for hybrid systems which have continuous, discrete and symbolic aspects.

The applications of this work are numerous: automatic inspection of mechanical or electronic components, reproduction of mechanical parts, etc. 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.

DEDS are usually modeled by finite state automata with partially observable events. Subsets of transitions can be disabled or enabled, depending on the application. Our approach is to use DEDS to drive a semi-autonomous visual sensing module which is capable of making decisions about the *state* of the exploration (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 exploration or move to a new mode of exploration.

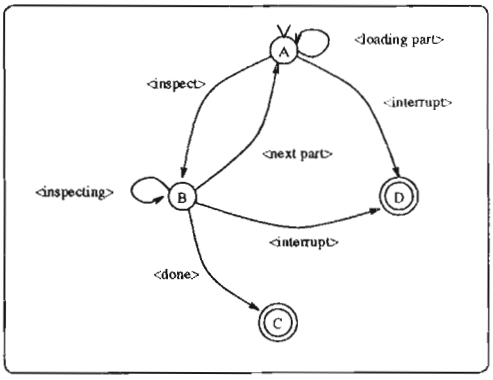


Figure 4: A Simple FSM

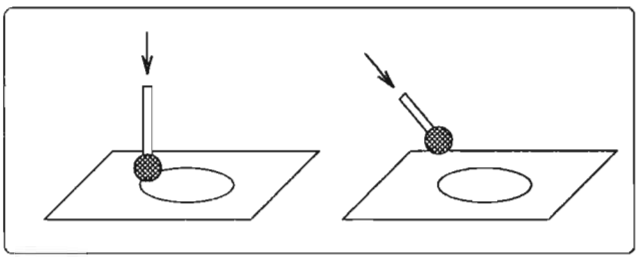


Figure 5: Bad Approach Vector

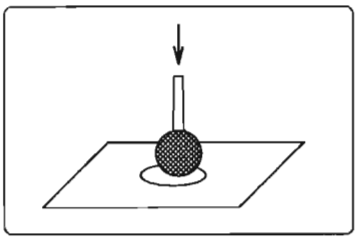


Figure 6: Probe diameter too large

3.1 Modeling an Observer

The tasks that the autonomous observer system executes can be modeled efficiently within a DEDS framework. We use the DEDS model 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.

In order to know the current state of the exploration process 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. The goal will be to make the system a strongly output stabilizable one 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.

Foveal and peripheral vision strategies could be used for the autonomous "focusing" on relevant aspects of the scene. Pyramid vision approaches and logarithmic sensors could be used to reduce the dimensionality and computational complexity for the scene under consideration.

3.2 Error States

We can utilize the observer framework for recognizing error states and sequences. The idea behind this recognition task is to be able to report on *visually incorrect* sequences. In particular, if there is a pre-determined observer model of a particular exploration 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 exploring robot so that it can correct its actions.

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

- occlusions between the observer camera and the part or probe.
- inappropriate approach vector position or orientation (see Figure 5).
- inappropriate probe size (see Figure 6).
- 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.

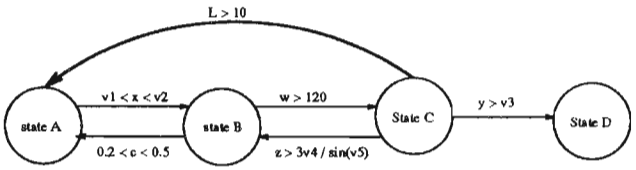
4 DRFSM

The Dynamic Recursive Context for Finite State Machines (DRFSM) is a new 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.

In our problem domain of machined parts, we can use DRFSM to exploit the recursive nature of many machined parts. Many machined features have similar exploration strategies. By using the same strategy for different features within a complicated part, we can reduce the number of control states needed to explore it to a manageable amount.

4.1 Definitions

- **Variable Transition Value:** Any variable value that depends on the level of recursion.
- **Variable Transition Vector:** The vector containing all variable transitions values, and is dynamically changed from level to level.
- **Recursive State:** A state calling another state recursively, and this state is responsible for changing the variable transition vector to its new value according to the new level.
- **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



trans. Variables	V1	V2	V3	V4	V5
Level 1	12	15	0.03	170	25
Level 2	10	12	0.07	100	35
Level 3	6	8	0.15	50	40

Figure 7: A Simple DRFSM

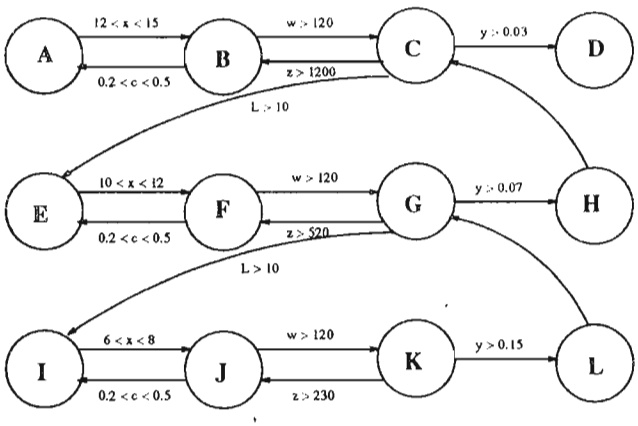


Figure 8: Flat Representation of a Simple DRFSM

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.

4.2 DRFSM Representation

We will use the same notation and terms of the ordinary FSMs, but some new notation to represent recursive states and variable transitions. First, we permit a new type of transition, as shown in Figure 7; (from state C to A), this is called the Recursive Transition (RT).

A recursive transition arrow (RTA) 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 one after changing the Variable Transition 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(parameter name). 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. Figure 8 is the equivalent FSM representation (or the flat representation) of the DRFSM shown in Figure 7, for three levels, and it illustrates the compactness and efficiency of the new notation for this type of process.

4.3 A Graphical DRFSM Interface

In developing the framework for reverse engineering, it has proven desirable to have a quick and easy means for modifying the DRFSM which drives the exploration 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[1]. It allows the user to graphically draw finite state machines, and output it 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 state machines.
- Modification of string parsing to accept recursive transition specification.
- Encoding of an event parser to prioritize incoming events from multiple sources.
- Implementation of the variable transition vector (VTV) acquisition (when making recursive transitions.)

The event parser was encoded to ensure that the automaton makes transitions on only one source of input. Each new event type requires the addition of a suitable event handler. 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 exploration utilities with no modifications. The code produced by the machine in Figure 9 was tested using a text interface before being linked with the rest of the experimental code.

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 exploration process without being familiar with the underlying code (given that all required user-defined events and modules are available).

5 Experiment

In conducting our experiments, we use a B/W CCD camera mounted on a Puma 560 robot arm (see Figure 10), and

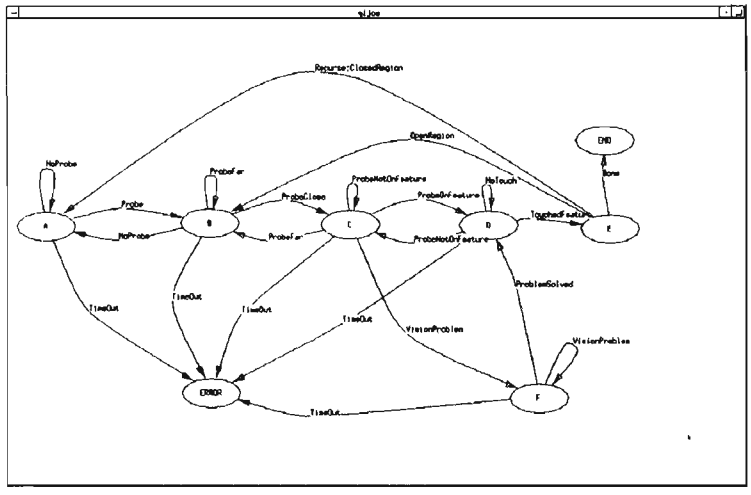


Figure 9: GIJoe Window w/DRFSM

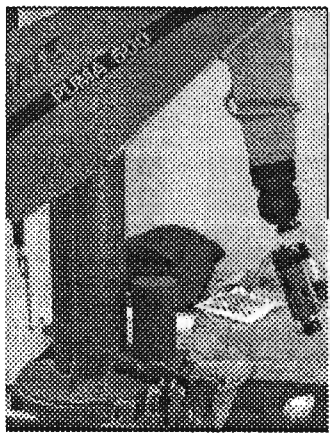


Figure 10: Experimental Setup

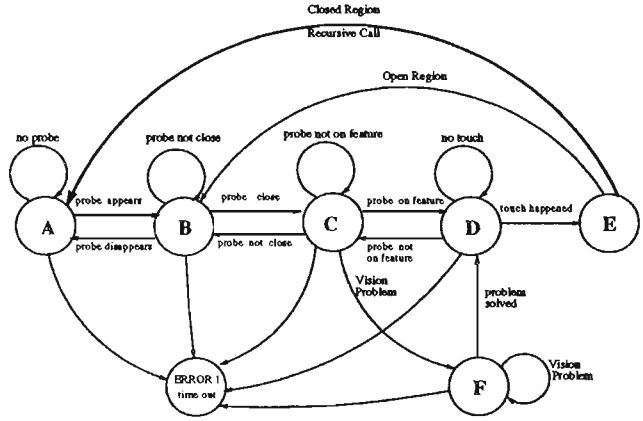


Figure 11: The DRFSM used in the experiment

simulate the operation of a CMM probe. Control signals that were generated by the DRFSM were converted to simple English commands and displayed to a human operator so that the simulated probe could be moved.

In order for the state machine to provide control, it must be aware of state changes in the system. As exploration takes place, the camera supplies images that are interpreted by a set of 2D and 3D vision processing algorithms and used to drive the DRFSM. These algorithms are described in greater detail in a technical report [4], 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. Our latest experiments used the robot and GIJoe-generated automata. One of them is described below.

The DRFSM generated by GIJoe is shown in figure 11. This machine has the following states:

- **A:** The initial state, waiting for the probe to appear.
- **B:** The probe appears, and waiting for it to be close.[1]

[1] "Close" is a relative measure of the distance between the

- **C:** Probe is close, but not on feature.
- **D:** The probe visually appears to be on feature but physical touch with the CMM machine has not occurred.
- **E:** Physical touch has happened. 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. Otherwise, the machine looks for another feature on the same level.
- **F:** This state is used 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.
- **ERROR1:** There is time limit for each part of this experiment to be done. If one of the modules does not finish within the limit, the machine will go to this state, which will report the error and terminate the experiment.

A part similar to the fuel pump cover from a Chevrolet engine was used in the experiment to test the exploration automaton. This piece offers interesting features and has a complex recursive structure. The piece was placed within view of the camera. Lighting in the room was adjusted so as to eliminate reflection and shadows on the part to be explored.

Some of the images from the experiment are shown in sequence in Figure 12.

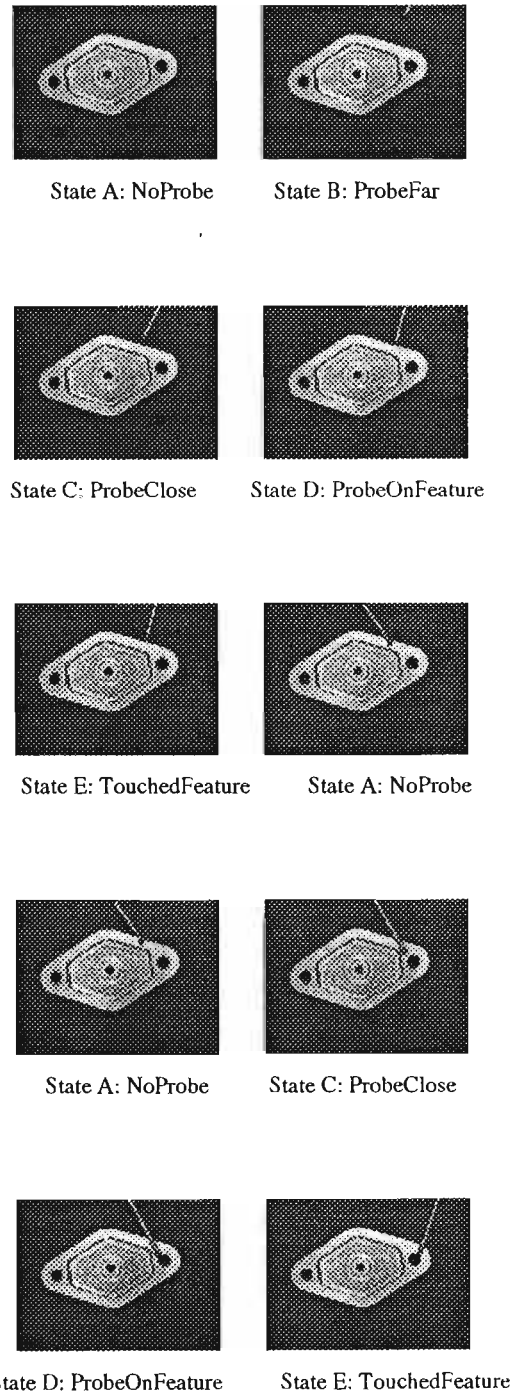


Figure 12: Cover Sequence

6 Conclusions

Discrete event dynamic system control has been explored for the reverse engineering problem. We have introduced a new context for use with problems which have a recursive aspect, such as machined parts. An interactive package has been developed which allows a user to graphically generate DRFSM automata. Experiments have been performed in the domain of inspection and reverse engineering where DRFSM provide robust control.

References

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

[2] NERODE, A., AND REMMEL, J. B. A model for hybrid systems. In *Proc. Hybrid Systems Workshop, Mathematical Sciences Institute* (May 1991). Cornell University.

[3] ÖZVEREN, C. M. *Analysis and Control of Discrete Event Dynamic Systems : A State Space Approach.* PhD thesis, Massachusetts Institute of Technology, August 1989.

[4] SOBH, T., OWEN, J., JAYNES, C., DEKHIL, M., AND HENDERSON, T. Intermediate results in active inspection and reverse engineering. C.S. Dept. UUCS-93-014, University of Utah, Salt Lake City, Utah, USA, June 1993.

probe and the current feature, and is specified using the VTV.