

Automated Inspection and Reverse Engineering

Tarek M. Sobh, Chris Jaynes,
Mohamed Dekhil, and Tom Henderson¹

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

Abstract. CAD/CAM (Computer Aided Design/Manufacturing) typically involves the design and manufacture of a mechanical part. The problem of reverse engineering is to take an existing mechanical part as the point of departure and to produce a design, and perhaps a manufacturing process, for the part. We explore a new approach to reverse engineering and inspection applications. Discrete event dynamic systems (DEDS) are investigated as a new framework for guiding and controlling the active exploration and perception of mechanical parts. The dynamic recursive context for finite state machines (DRFSM) is introduced as a new DEDS tool for utilizing the recursive nature of the mechanical parts under consideration. The framework utilizes DRFSM DEDS for constructing a goal-directed autonomous observer for inspection and reverse engineering purposes. The developed techniques are then used to aid CAD pre-constructed models for inspection purposes and/or to develop models for subsequent manufacturing, thus, closing the sensing \rightarrow modeling \rightarrow manufacturing \rightarrow inspection loop.

Keywords. Automation, Computer Vision, Control Systems, Discrete Event Dynamic Systems, Inspection, Intelligent Systems, Manufacturing, Reverse Engineering, Robotics.

1 Introduction

Developing frameworks for inspection and reverse engineering applications is an essential activity in many engineering disciplines. Usually, too much time

¹This work was supported in part by DARPA grant N00014-91-J-4123 and NSF grant CDA 9024721. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

is spent in designing hardware and software environments, in order to be able to attack a specific problem. One of the purposes of this work is to provide a basis for solving a class of inspection and reverse engineering problems.

CAD/CAM (Computer Aided Design, Manufacturing) typically involves the design and manufacture of a mechanical part. The problem of reverse engineering is to take an existing mechanical part as the point of departure and to inspect or produce a design, and perhaps a manufacturing process, for the part. The techniques that we explore can hopefully be used for a variety of applications. We use an observer agent to sense the current world environment and make some measurements, then supply relevant information to a control module that will be able to make some design choices that will later affect manufacturing and/or inspection activities. This involves both autonomous and semi-autonomous sensing.

We use a recursive dynamic strategy for exploring machine parts. A discrete event dynamic system (DEDS) [7,16,17,23,24,27] framework is designed for modeling and structuring the sensing and control problems. The dynamic recursive context for finite state machines (DRFSM) is introduced as a new DEDS tool for utilizing the recursive nature of the mechanical parts under consideration. This paper describes what this means in more detail.

Next, we discuss the objectives and research questions. Then we discuss the methodology and proceed to describe some experiments and results. We conclude by detailing the current developments and the integration efforts.

2 Objectives and Research Questions

The objective of this research project is to explore the basis for a consistent software and hardware environment, and a flexible system that is capable of performing a variety of inspection and reverse engineering activities. In particular, we concentrate on the adaptive automatic extraction of some properties of the world to be sensed and on the subsequent use of the sensed data for producing reliable descriptions of the sensed environments for manufacturing and/or description refinement purposes.

We use an observer agent with some sensing capabilities (vision and touch) to actively gather data (measurements) of mechanical parts. Geometric descriptions of the objects under analysis are generated and expressed in terms of a Computer Aided Design system. The geometric design is then used to construct a prototype of the object. The manufactured prototype are then inspected and compared with the original object using the sensing interface and refinements made as necessary.

DEDS can be simply described as:

Dynamic systems (typically asynchronous) in which state transitions are triggered by discrete events in the system.

It is possible to *control* and *observe* hybrid systems (systems that involve

continuous, discrete and symbolic parameters) under uncertainty using DEDS formulations [16,20,21,22,26,30].

The applications of this work are numerous : e.g., automatic inspection of mechanical or electronic components and reproduction of mechanical parts. Moreover, the experience gained in performing this research allows us to study the subdivision of the solution into reliable, reversible, and an easy-to-modify software and hardware environments.

3 Methodology for Inspection and Reverse Engineering

In this section we describe the solution methodology and discuss the components separately. The control flow is described and the methods, specific equipment and procedures are also discussed in detail.

We use a vision sensor (B/W CCD camera) and a coordinate measuring machine (CMM) with the necessary software interfaces to a Sun Sparcstation as the sensing devices. The object is to be inspected by the cooperation of the observer camera and the probing CMM. A DEDS is used as the high-level framework for exploring the mechanical part. A dynamic recursive context for finite state machines (DRFSM) is used to exploit the recursive nature of the parts under consideration.

3.1 Discrete Event Dynamic Systems

DEDS are usually modeled by finite state automata with partially observable events together with a mechanism for enabling and disabling a subset of state transitions [3,18,20,21,22,23,24]. We propose that this model is a suitable framework for many reverse engineering tasks. In particular, we use the model as a high-level structuring technique for our system.

We advocate an approach in which a stabilizable semi-autonomous visual sensing interface would be capable of making decisions about the *staté* of the observed machine part and the probe. Thus providing both symbolic and parametric descriptions to the reverse engineering and/or inspection control module. The DEDS-based active sensing interface will be discussed in the following section.

3.1.1 Modeling and Constructing 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 [26]. 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.

3.1.2 Error States and Sequences

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 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 an operator or autonomously supply feedback to the inspecting robot so that it could correct its actions. An example of errors in inspection is unexpected occlusions between the observer camera and the inspection environment, or probing the part in a manner that might break the probe. 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.

3.1.3 Hierarchical Representation

Figure 1 shows a hierarchy of three submodels. Motives behind establishing hierarchies in the DEDS modeling of different exploration tasks includes reducing the search space of the observer and exhibiting modularity in the controller design. This is done through the designer, who subdivides the task space of the exploring robot into separate submodels that are inherently independent. Key events cause the transfer of the observer control to new submodels within the hierarchical description. Transfer of control through

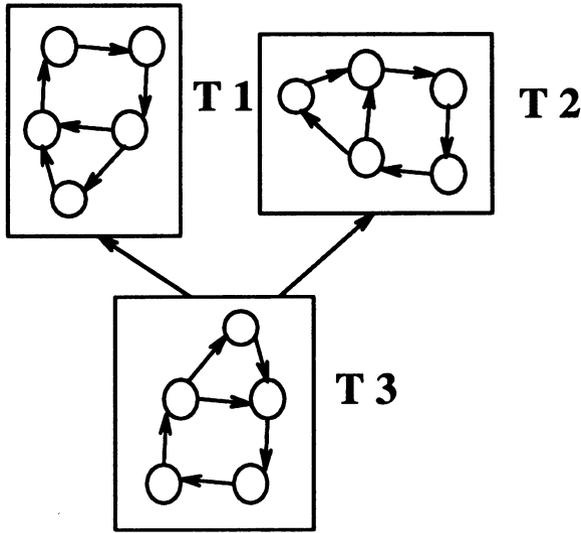


Figure 1: A Hierarchy of Tasks.

the observer hierarchy of models allows coarse to fine shift of attention in recovering events and asserting state transitions.

3.1.4 Mapping Module

The object of having a mapping module is to dispense with the need for the manual design of DEDS automaton for various platform tasks. In particular, we would like to have an off line module which is to be supplied with some symbolic description of the task under observation and whose output would be the code for a DEDS automata that is to be executed as the observer agent. The problem reduces to figuring out what is an appropriate form for the task description. The error state paradigm motivated regarding this problem as the inverse problem of determining acceptable languages for a specific DEDS observer automaton. In particular, we suggest a skeleton for the mapping module that transform a collection of input strings into an automaton model.

The idea is to supply the mapping module with a collection of strings that represents possible state transition sequences. The input highly depends on the task under observation, what is considered as relevant states and how coarse the automaton should be. The sequences are input by an operator. It should be obvious that the “Garbage-in-garbage-out” principle holds for the construction process; in particular, if the set of input strings is not representative of all possible scene evolutions, then the automaton would be a faulty one. The experience and knowledge that the operator have would influence the outcome of the resulting model. However, it should be noticed that the

level of experience needed for providing these sets of strings is much lower than the level of experience needed for a designer to actually construct a DEDS automaton manually. The description of the events that cause transitions between different symbols in the set of strings should be supplied to the module in the form of a list.

As an illustrative example, suppose that the task under consideration is simple grasping of one object and that all we care to know is three configurations; whether the hand is alone in the scene, whether there is an object in addition to the hand and whether enclosure has occurred. If we represent the configurations by three states h , h_o and h_c , then the operator would have to supply the mapping module with a list of strings in a language, whose alphabet consists of those three symbols, and those strings should span the entire language, so that the resulting automaton would accept all possible configuration sequences. The mapping from a set of strings in a regular language into a minimal equivalent automaton is a solved problem in automata theory.

One possible language to describe this simple automaton is :

$$L = hh^*h_o h_o^*h_c h_c^*$$

and a corresponding DEDS automaton is shown in Figure 2.

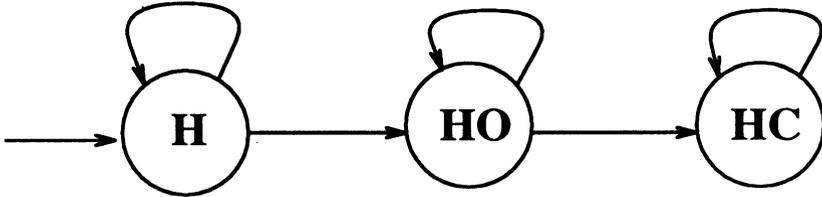


Figure 2: An Automaton for Simple Grasping.

The best-case scenario would have been for the operator to supply exactly the language L to the mapping module with the appropriate event definitions. However, it could be the case that the set of strings that the operator supplies do not represent the task language correctly, and in that case some learning techniques would have to be implemented which, in effect, augment the input set of strings into a language that satisfies some pre-determined criteria. For example, y^* is substituted for any string of y 's having a length greater than n , and so on. In that case the resulting automaton would be correct up to a certain degree, depending on the operator's experience and the correctness of the learning strategy.

3.2 Sensing Strategy

We use a B/W CCD camera and a coordinate measuring machine (CMM) to sense the mechanical part. A DRFSM implementation (see below) of a

discrete event dynamic system (DEDS) algorithm is used to facilitate the state recovery of the inspection process. DEDS are suitable for modeling robotic observers as they provide a means for tracking the *continuous*, *discrete* and *symbolic* aspects of the scene under consideration [3,20,21]. Thus the DEDS controller will be able to *model* and *report* the state evolution of the inspection process.

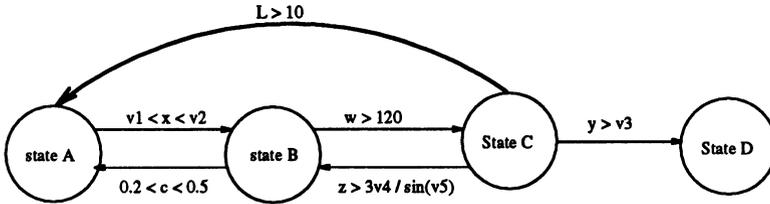
In inspection, the DEDS guides the sensing machines to the parts of the objects where discrepancies occur between the real object (or a CAD model of it) and the recovered structure data points and/or parameters. The DEDS formulation also compensates for noise in the sensor readings (both ambiguities and uncertainties) using a probabilistic approach for computing the 3-D world parameters [26]. The recovered data from the sensing module is then used to drive the CAD module. The DEDS sensing agent is thus used to collect data of a *passive* element for designing *structures*; an exciting extension is to use a similar DEDS observer for moving agents and subsequently design *behaviors* through a learning stage.

3.3 The Dynamic Recursive Context for Finite State Machines

The Dynamic Recursive Context for Finite State Machines (DRFSM) is a new methodology to represent and implement multi-level recursive processes using systematic implementation techniques. By multi-level process we mean any processing operations that are done repetitively with different parameters. DRFSM has proved to be a very efficient way to solve many complicated problems in the inspection paradigm using an easy notation and a straight forward implementation, especially for objects that have similar multi-level structures with different parameters. The main idea of the DRFSM is to reuse the conventional DEDS Finite State Machine for a new level after changing some of the transition parameters. After exploring this level, it will retake its old parameters and continue exploring the previous levels. Also, the implementation of such machines can be generated automatically by some modification to an existing reactive behavior design tool called GIJOE [4] that is capable of producing code from state machine descriptions (drawings) by adding a recursive representation to the conventional representation of finite state machines, and then generating the appropriate code for it.

3.3.1 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.



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 3: A Simple DRFSM

- **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 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.

3.3.2 DRFSM Representation

We will use the same notation and terms of the ordinary FSM's, but some new notation to represent recursive states and variable transitions. First, we permit a new type of transition, as shown in Figure 3; (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 $V(\text{parameter name})$. All variable parameters of all state transitions constitute the Variable Transition Vector. Figure 4 is the equivalent FSM representation (or the flat representation) of the DRFSM shown in Figure 3, for three levels, and it illustrates the compactness and efficiency of the new notation for this type of process. In many cases, however,

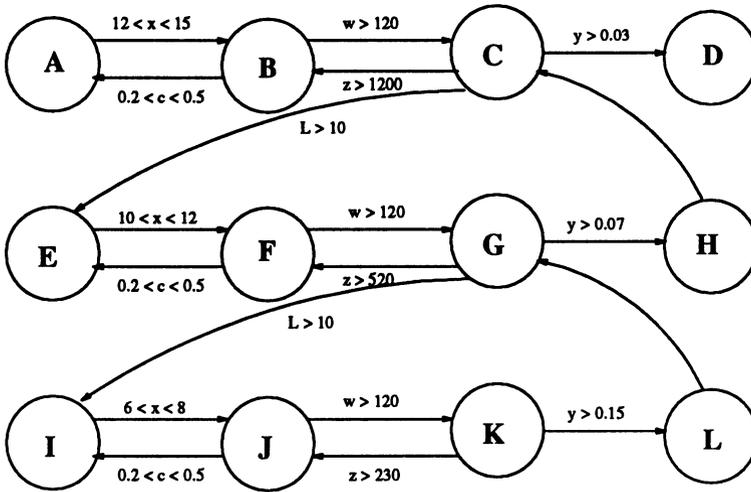


Figure 4: Flat Representation of a Simple DRFSM

it is impossible to build the equivalent FSM for a process because some values of its Variable Transition Vector are undefined until their corresponding level is reached. In these cases DRFSM's are the most appropriate way to deal with such applications.

3.3.3 Implementation of DRFSM

We intend to develop extensions to GIJOE [4] by adding some facilities to allow drawing of DRFSM's and to generate the appropriate C code with a recursive call to some states with variable transition conditions. The required modifications will be accomplished in two phases:

- Drawing Phase.
- Code Generation Phase.

In the drawing phase a new arrow will be added (RTA) to represent a recursive call to any state. Also a notation for variable transition value will be added as shown in Figure 5.

In the code generation phase, it is very important to preserve backward compatibility; fortunately, that is easy since we can check for the existence of RTA's. If no RTA is found, then it is a FSM and the code generated for this machine will be the same as before. On the other hand, if any RTA is found, then the following steps are required:

- Collect all variable transitions to form the VTV.

VTV : (v1, v2, v3, v4, v5)

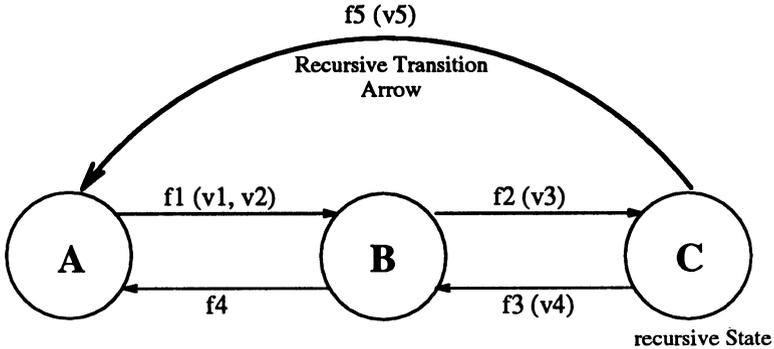


Figure 5: New Notation for GIJOE

- For each RTA in the figure build a user-defined function: `Get_New_VTV` to be filled by the user of GIJOE later, since this function is very application dependent, then its purpose is to get the values of the new vector to be used in the new level of recursion, and it will be called from the recursive state.
- All states' functions will have a parameter which is the VTV.

With these modifications backward compatibility is guaranteed and the implementation of any DRFSM is easily maintained. In Appendix A, a generated code for a DRFSM is shown.

3.3.4 How to use DRFSM ?

To apply DRFSM for any problem the following steps are required:

- **Problem Analysis:** Divide the problem into states, so that each state accomplishes a simple task.
- **Transition Conditions:** Find the transition conditions between the different states.
- **Explore the repetitive part in the problem (recursive property) and specify the recursive states.** Some problems however may not have this property. In those cases a FSM is a better solution.
- **VTV formation:** If there are different transitions values for each level; these variables have to be defined.
- **Error trapping:** Using robust analysis, a set of possible errors can be established; then one or more Dead-End state(s) are added.

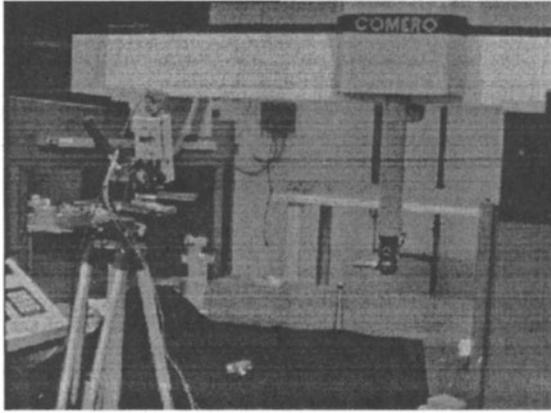


Figure 6: Experimental Setup

- **DRFSM Design:** Use GIJOE to draw the DRFSM and generate the corresponding C code.
- **Implementation:** The code generated by GIJOE has to be filled out with the exact task of each state, the error handling routines should be written, and the required output has to be implemented as well.

3.3.5 Applying DRFSM in Feature extraction

An experiment was performed for inspecting a mechanical part using a camera and the coordinate measuring machine. A predefined DRFSM state machine was used as the observer agent skeleton. The camera was placed on a stationary tripod at the base of the table so that the part was always in view. The probe could then extend into the field of view and come into contact with the part, as shown in Figure 6.

Symbolic Representation of Features: For the above experiment we were concerned with open regions (O) and closed regions (C). Any closed region may contain other features (the recursive property). Using parenthesis notation the syntax for representing features can be written as follow:

< feature > :: C(< subfeature >) | C()
 < subfeature > :: < term >, < subfeature > | < term >
 < term > :: O | < feature >

For example, the symbolic notation of Figure 7 is

C(O,C(O,C()),C(O)),C()

Figure 8 shows the graphical representation of this recursive structure which is a tree-like structure. Future modifications to DRFSM's includes allowing different functions for each level.

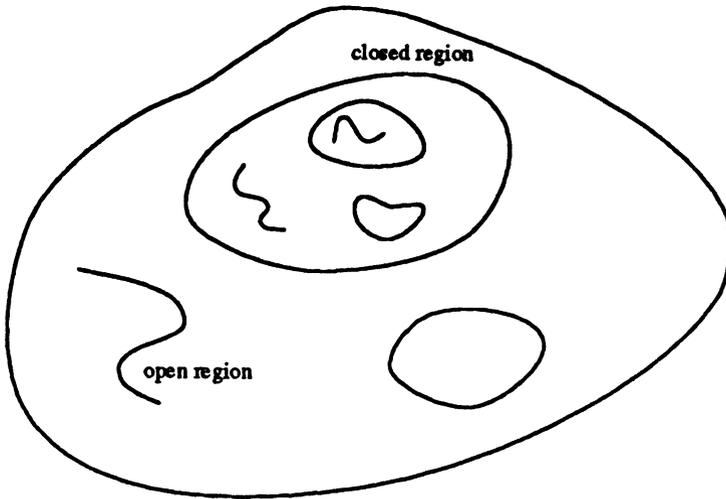


Figure 7: An Example for a Recursive Object

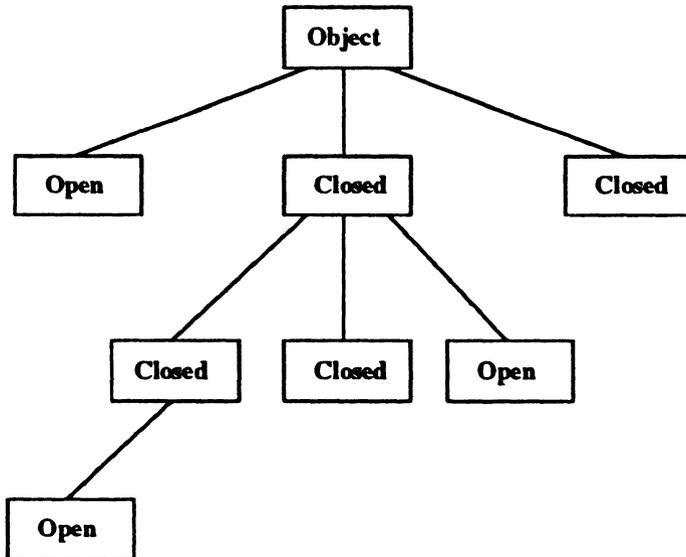


Figure 8: Graph for the Recursive Object

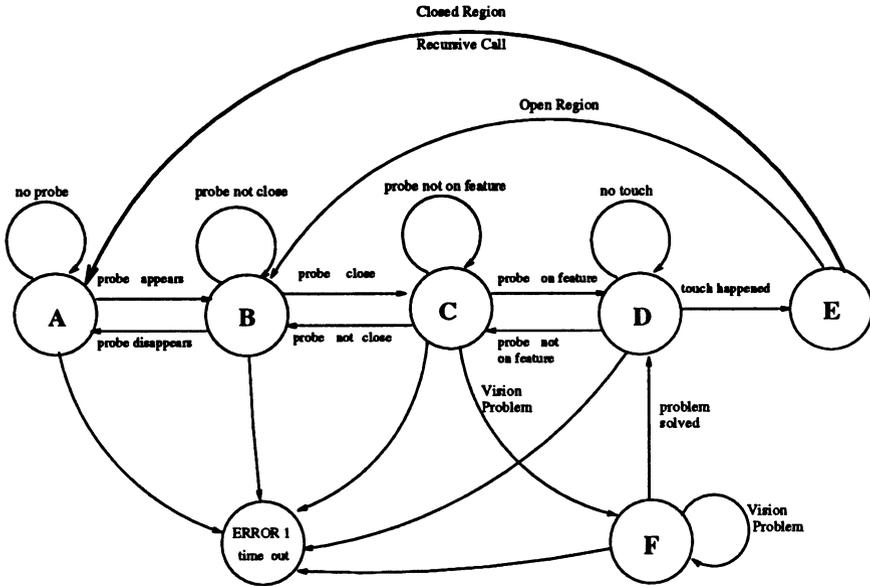


Figure 9: A DRFSM DEDS for Inspection

Figure 9 shows a simple DRFSM DEDS machine for the exploration and inspection of mechanical parts, using both active vision and touch sensors.

4 Visual Processing

In order for the state machine to work, it must be aware of state changes in the system. As inspection takes place, the camera supplies images that are interpreted by a vision processor and used to drive the DRFSM.

The vision processor provides two separate pieces of information that are required by the machine, intrinsic information about the part to be inspected, and state information as the inspection takes place.

4.1 Extracting Feature Information

The state machine requires information about the 2-D features on the part to be inspected. We divide 2-D features into two categories, *open features* and *closed features*. An open feature is considered to be an edge that, when followed, has no closure. Closed features have the property that, through an edge search, we are able to complete a closed loop.

After digitizing an image of the part, edge responses are captured using the zero-crossing technique. Next, we search the edge responses for the feature type that they represent. Using a recursive search and the orientation

information given by the zero-crossing algorithm, we are able to label each edge as part of a closed or open feature.

4.2 Deciding Feature Relationships

Once we have found all of the features, we now search for the relationships between them. In the final representation of intrinsic information about the part, it is important to know which feature lies “within” another closed feature.

Consider a scene with two features, a part with an external boundary and a single hole. We would like to represent this scene with the string: “C(C())”. This can be interpreted as, a closed region within another closed region.

4.3 Visual Observation of States

The visual processor supplies the proper input signals to the DRFSM DEDS as the inspection takes place. These signals are dependent upon the state of the scene and are triggered by discrete events that are observed by the camera.

The visual processor layer is made up of several filters that are applied to each image as it is captured. Several pieces of information must be known about the scene before a signal is produced. The location of the part, the location of the probe, the distance between them, the number of features on the part, and the distance to the closest feature.

Once this information is known, we are able to supply the correct signal that will drive the DRFSM DEDS. The machine, will then switch states appropriately and wait for the next valid signal. This process is a recursive one, in that, the machine will be applied recursively to the closed features. As the probe enters a closed region, another machine will be activated, that will inspect the smaller closed region with the same strategy that was used on the enclosing region.

4.3.1 Constructing the Recursive Relation

One of the problems we have encountered was converting the set of relations between closed regions to the proposed syntax for describing objects. For example, the syntax of Figure 10 is:

$$C(C(C()),C()),C())$$

and the relations generated by the image processing program are:

Relation (1): $B \subset A$

Relation (2): $C \subset A$

Relation (3): $D \subset B$

Relation (4): $D \subset A$

Relation (5): $E \subset B$

Relation (6): $E \subset A$

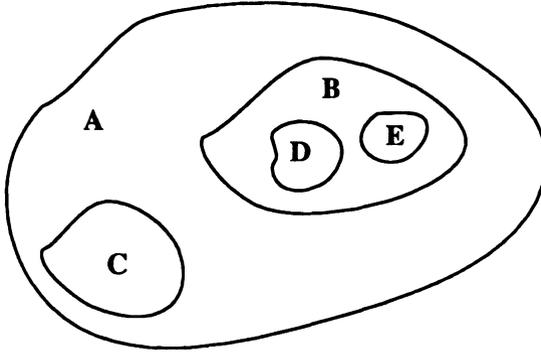


Figure 10: A Hierarchy Example

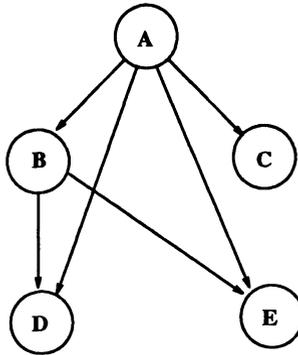


Figure 11: The graph associated with the example

These relations can be represented by a graph as shown in Figure 11. The problem is to convert this graph to an equivalent tree structure, which is the most convenient data structure to represent our syntax. Our method is to scan the relations, count the number of occurrences for each closed region name mentioned in the left side of the relations giving an array $RANK(x)$, where $x \in \{A, B, C, \dots\}$, and select the relations $(x_1 \subset x_2)$ that satisfy the following condition:

$$RANK(x_1) - RANK(x_2) = 1$$

This guarantees that no redundant relations will be selected. Applying this algorithm to the relations of Figure 10 we have,

$$RANK(A) = 0; RANK(B) = 1; RANK(C) = 1;$$

$$RANK(D) = 2; RANK(E) = 2$$

The selected relations will be:

$$B \subset A; C \subset A; D \subset B; E \subset B$$

Now arranging these relations to construct the syntax gives:

$$A(B()) \longrightarrow A(B(), C()) \longrightarrow A(B(D()), C()) \longrightarrow A(B(D(), E()), C())$$

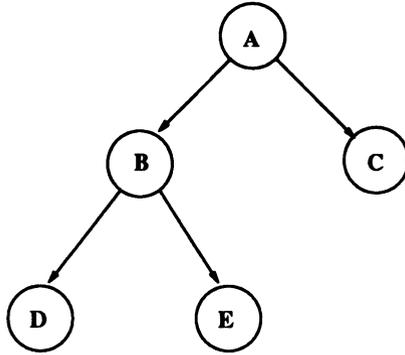


Figure 12: The tree associated with the example

which is the required syntax. A tree representing this syntax is easily constructed and shown in Figure 12. The next step would be to insert the open regions, if any, and this is done by traversing the tree from the maximum depth and upwards. Any open region can be tested by checking any point in it and checking whether it lies within the maximum depth leaves of the closed region's tree hierarchy. (The test is easily done by extending a line and checking how many times it intersects a closed region, as in the test for closed regions enclosures.) Then the upper levels of the hierarchy are tested in ascending order till the root is reached or all open regions have been exhausted. Any open region found to be inside a closed one while traversing the tree is inserted in the tree as a child for that closed region. It should be noticed that this algorithm is *not* a general graph to tree conversion algorithm; it only works on the specific kind of graphs that the image processing module recovers. That is, the conversion algorithm is *tailored* to the visual recursion paradigm.

5 Experiments and Results

An experiment was performed that integrated the visual system with the state machine. An appropriate DRFSM was generated by observing the part and generating the feature information. A mechanical part was put on a black velvet background on top of the coordinate measuring machine table to simplify the vision algorithms. The camera was placed on a stationary tripod at the base of the table so that the part was always in view. The probe could then extend into the field of view and come into contact with the part, as shown in Figure 6.

Once the first level of the DRFSM was created, the experiment could proceed as follows. First, an image was captured from the camera. Next, the appropriate image processing takes place to find the position of the part,

the number of features observed (and the recursive string), and the location of the probe. A program using this information produces a state signal that is appropriate for the scene. The signal is read by the state machine and the next state is produced and reported. Each closed feature is treated as a recursive problem, as the probe enters a closed region, a new level of the DRFSM is generated with a new transition vector. This new level then drives the inspection for the current closed region.

5.1 DRFSM DEDS example

The specific dynamic recursive DEDS automaton generated for the test was a state machine G . Where $X = \{\text{Initial}, \text{EOF}, \text{Error}, \text{A}, \text{B}, \text{C}, \text{D}\}$ and $\Sigma = \{1, 2, 3, 4, 5, 6, 7, 8, 9, \text{eof}\}$. The state transitions were controlled by the input signals supplied by intermediate vision programs. There are four stable states A, B, C, and D that describe the state of the probe and part in the scene. The three other states, Initial, Error, and EOF specify the actual state of the system in special cases. The states can be interpreted as:

- Initial State: Waiting for first input signal
- A: Part Alone in Scene
- B: Probe and Part in Scene, probe is far from part.
- C: Probe and Part in Scene, probe is close to part.
- D: Probe touching or overlapping part. (recursive state)
- Error: An invalid signal was received.
- EOF: The End of File signal was received.

5.2 Results

Two typical sequences from a probing task were run. In the first sequence (Figure 14), the probe was introduced into the scene and moved in a legal way (accepted by stable states in the machine) towards the part until contact was made. Next, the probe backed off and again approached until the probe and part overlapped. The automaton was forced into an error state by approaching from the other side of the part much too fast. The probe was not seen until it was too close to the object body. Because a transition from state A to C is invalid, an error state is reached. The part used was a simple one with only one hole, that is, it is represented by : C(C()).

Another sequence was tried out (Figure 15), the part was more complex, the representation was recovered to be the following string : C(C(),C(C()),C()). The probe was introduced into the scene and moved legally towards the part.

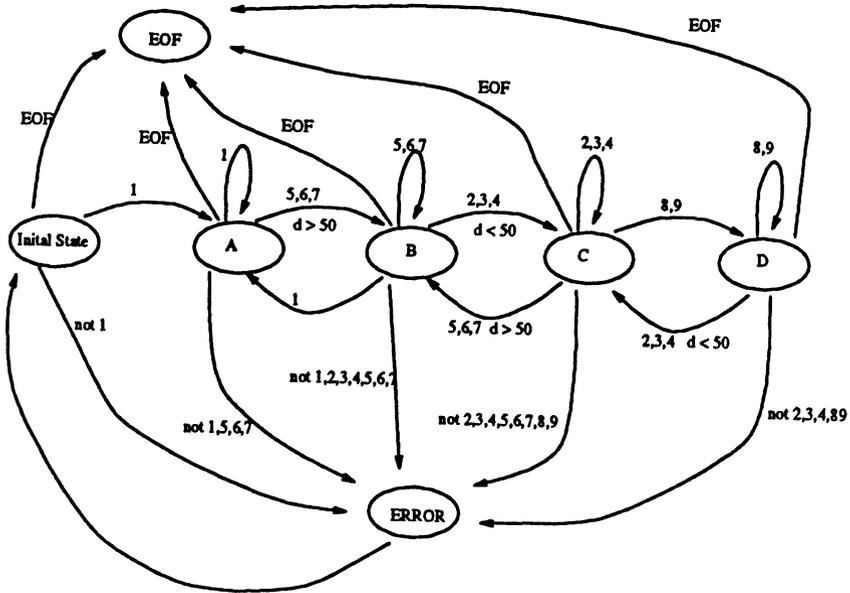


Figure 13: State Machine Used in Test

Next, the probe backed off and again approached until the probe and the part overlapped. The automaton was forced into an error state by the sudden disappearance of the probe after it was very close to the part. Because a transition from state C to state A is invalid, an error state is reported. Each image was displayed on a terminal window as it was captured along with the corresponding state of the automaton. The same state representations are displayed for different layers in the DRFSM (i.e. for different features).

6 Integration and Current Developments

The application environment we eventually intend to develop consists of three major working elements: the sensing, design, and manufacturing modules. The ultimate goal is to establish a computational framework that is capable of deriving designs for machine parts or objects, inspect and refine them, while creating a flexible and consistent engineering environment that is extensible. The control flow is from the sensing module to the design module and then to the manufacturing component. Feedback can be re-supplied to the sensing agent to inspect manufactured parts, compare them to the originals and continue the flow in the loop until a certain tolerance is met. The system is intended to be ultimately as autonomous as possible. We intend to study what parts of the system can be implemented in hardware. Some parts seem to be inherently suited to hardware, which will be discussed later, while some other parts of the system may be possible to put in hardware, but

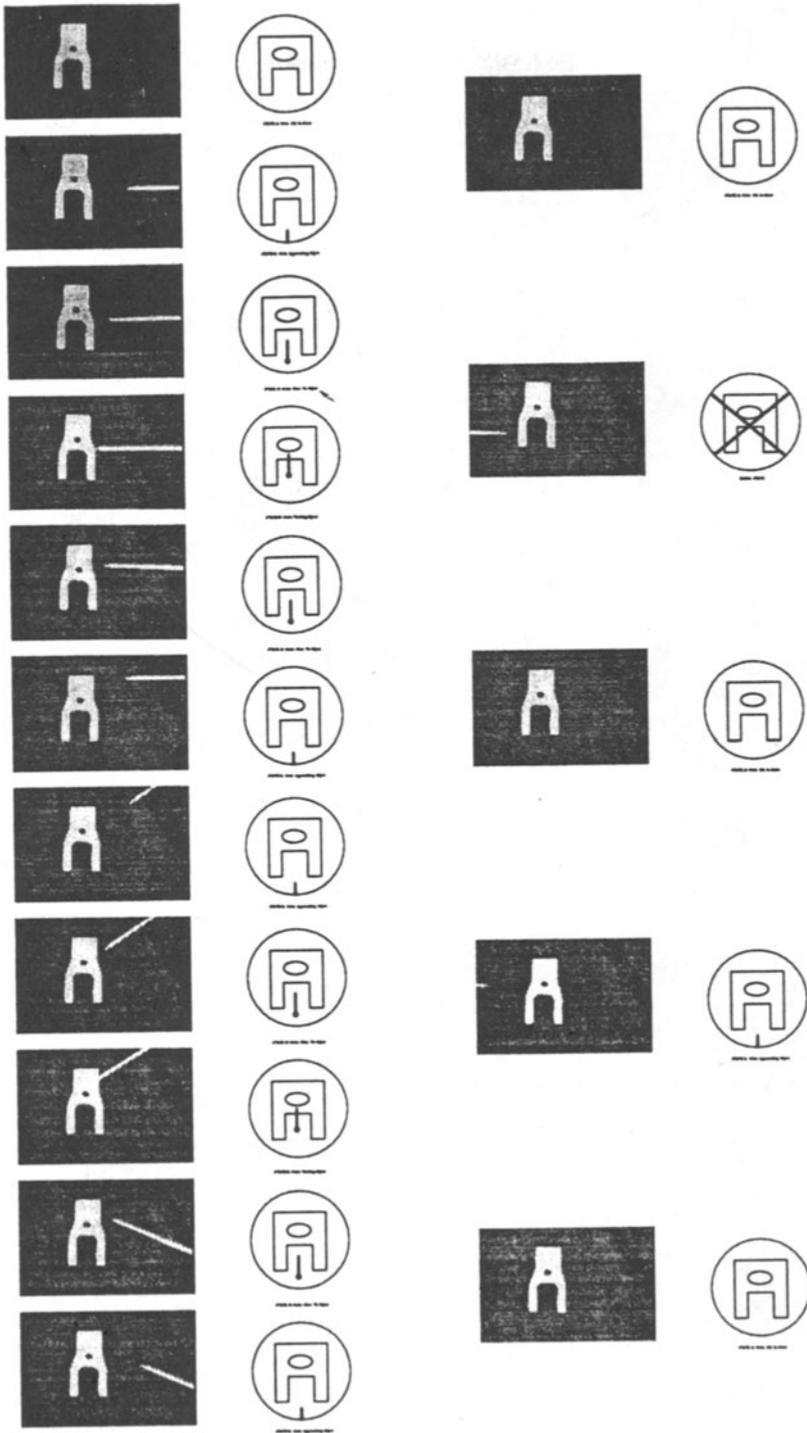


Figure 14: Test Sequence (1)

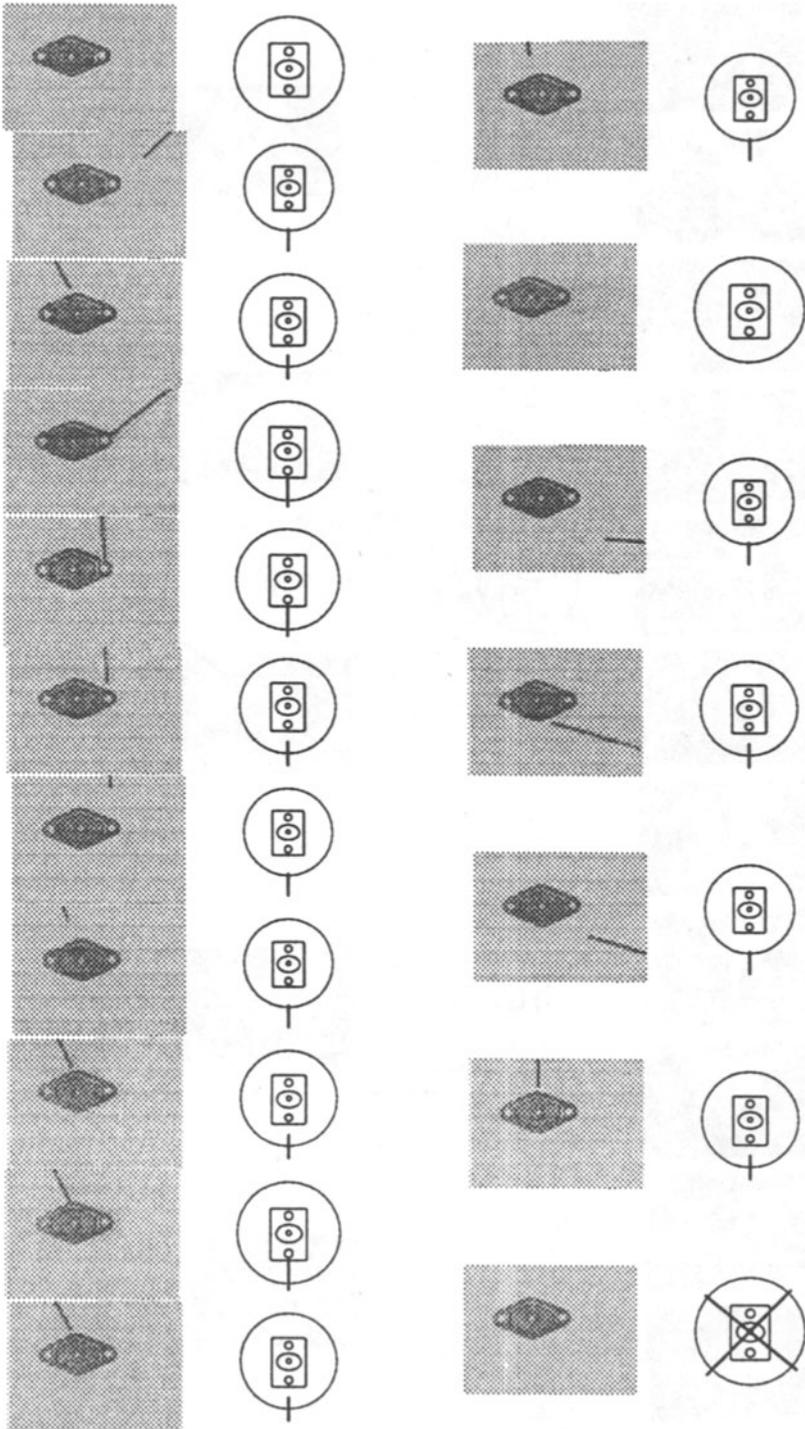


Figure 15: Test Sequence (2)

experimentation will provide the basis for making that decision. Providing language interfaces between the different components in the inspection and reverse engineering control loop is an integral part of the project.

6.1 Robotics and Sensing

We shall be using a robot arm (a PUMA 560), a vision sensor (B/W CCD camera) mounted on the end effector and a coordinate measuring machine (CMM) with the necessary software interfaces to a Sun Sparcstation as the sensing devices. A DRFSM DEDS algorithm will be used to coordinate the movement of the robot sensor and the CMM. We have some experience in formulating observers using DEDS [26]. DEDS are suitable for modeling robotic observers as they provide a means for tracking the *continuous*, *discrete* and *symbolic* aspects of the scene under consideration [3,20,21]. Feedback will be provided to the robot arm, based on visual observations, so that the object(s) under consideration can be explored [8,14,26]. The DEDS control algorithm will also guide the CMM to the relevant parts of the objects that need to be explored in more detail (curves, holes, complex structures, etc). Thus the DEDS controller will be able to *model*, *report*, and guide the robot and the CMM to reposition *intelligently* in order to recover the structure and shape parameters.

6.2 Computer Aided Design and Manufacturing

The data and parameters derived from the sensing agent are then to be fed into the CAD system for designing the geometry of the part(s) under inspection. We intend to use the α_1 design environment [2,11,25,29] for that purpose. The goal is to provide automatic programming interfaces from the data obtained in the sensing module to the α_1 programming environment. The parametric and 3-D point descriptions are to be integrated to provide consistent and efficient surface descriptions for the CAD tool. For pure inspection purposes the computer aided geometric description of parts could be used as a *driver* for guiding both the robotic manipulator and the coordinate measuring machine for exploring the object and recognizing discrepancies between the real part and the model.

The computer aided design parameters are then to be used for manufacturing the prototypes. Considerable effort has been made for automatically moving from a computer aided geometric model to a process plan for making the parts on the appropriate NC machines and then to automatically generate the appropriate machine instructions [9,10,12]. We intend to use the Monarch VMC-45 milling machine as the manufacturing host. The α_1 system will produce the NC code for manufacturing the parts.

6.3 VLSI and Languages

The software and hardware requirements of the environment are the backbone for this project. We intend to select parts of the system implementation and study the possibility of hardwiring them. There has been considerable effort and experience in VLSI chip design [6,13] and one of the sub-problems would be to study the need and efficiency of making customized chips. The DEDES model, as an automaton, is very suitable for Path Programmable Logic (PPL) implementation. A number of the visual sensing algorithms could be successfully implemented in PPL, saving considerable computing time. Integrated circuits for CAGD surface manipulation is an effort that is already underway. We intend to investigate a new area: the possibility of implementing the DEDES part of the system in integrated circuitry.

There is a lot of interfacing involved in constructing the inspection and reverse engineering environments under consideration. Using multi-language object-based communication and control methodology between the three major components (Sensing, CAD and CAM) is essential. We intend to use a common shared database for storing data about the geometric model and the rules governing the interaction of the different phases in the reproduction and inspection paradigms [19,28].

7 Conclusions

We propose a new strategy for inspection and/or reverse engineering. We concentrate on the inspection of machine parts. We also describe a framework for constructing a full environment for generic inspection and reverse engineering. The problem is divided into *sensing*, *design*, and *manufacturing* components with an underlying software and hardware backbone. This project aims at developing automated control strategies for sensing the world and coordinating the activities between the different phases in the environment. We use a recursive DEDES DRFSM framework to construct an intelligent module for inspection and reverse engineering. The developed framework utilizes existing knowledge to formulate an adaptive and goal-directed strategy for exploring mechanical parts.

References

- [1] R. Bajcsy, "Active Perception," *Proceedings of the IEEE*, Vol. 76, No. 8, August 1988.
- [2] M. J. Banks and E. Cohen, "Realtime B-Spline Curves from Interactively Sketched Data," *Proceedings of the 1990 Symposium on Interactive 3-D graphics*, ACM, March 1990.
- [3] A. Benveniste and P. L. Guernic, "Hybrid Dynamical Systems Theory and the SIGNAL Language," *IEEE Transactions on Automatic Control*, Vol. 35, No. 5, May 1990.
- [4] M. J. Bradakis, "Reactive Behavior Design Tool," Master's Thesis, Computer Science Department, University of Utah, January 1992.
- [5] Y. Brave and M. Heymann, *Control of Discrete Event Systems Modeled as Hierarchical State Machines*, Technical Report CIS-9012, Computer Science Department, TECHNION - Israel Institute of Technology, March 1991.
- [6] T. M. Carter, K. F. Smith, S. R. Jacobs, and R. M. Neff, "Cell Matrix Methodologies for Integrated Circuit Design," *Integration, The VLSI Journal*, 9(1), 1990.
- [7] C. Chase, J. Serrano, and P. Ramadge, *Periodicity and Chaos from Switched Flow Systems : Contrasting Examples of Discretely Controlled Continuous Systems*, Technical Report, Department of Electrical Engineering, Princeton University, January 1991.
- [8] F. Chaumette and P. Rives, "Vision-Based-Control for Robotic Tasks," In *Proceedings of the IEEE International Workshop on Intelligent Motion Control*, Vol. 2, pp. 395-400, August 1990.
- [9] J. J. Chou and E. Cohen, "Computing Offsets and Tool Paths with Vornoi Diagrams," Technical Report UUCS-89-017, Department of Computer Science, University of Utah, 1989.
- [10] J. J. Chou and E. Cohen, "Constant Scallop Height Tool Path Generation," Technical Report UUCS-89-011, Department of Computer Science, University of Utah, 1989.
- [11] E. Cohen, T. Lyche, and R. F. Riesenfeld, "Discrete B-Splines and Subdivision Techniques in Computer Aided Geometric Design and Computer Graphics," *Computer Graphics and Image Processing*, 14(2):87-111, October 1980.
- [12] S. Drake and S. Sela, "A Foundation for Features," *Mechanical Engineering*, 111(1), January 1989.

- [13] J. Gu and K. Smith, "A Structured Approach for VLSI Circuit Design," *IEEE Computer*, 22(11), 1989.
- [14] C. D. Hansen and T. C. Henderson, "CAGD-Based Computer Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(10) : 1181-1193, 1989.
- [15] M. Heymann, "Concurrency and Discrete Event Control", In *Proceedings of the IEEE Conference on Decision and Control*, December 1989.
- [16] Y. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems", *IEEE Transactions on Automatic Control*, July 1987.
- [17] K. Inan and P. Varaiya, "Finitely Recursive Process Models for Discrete Event Systems", *IEEE Transactions on Automatic Control*, Vol. 33, No. 7, July 1988.
- [18] Y. Li and W. M. Wonham, "Controllability and Observability in the State-Feedback Control of Discrete-Event Systems", *Proc. 27th Conf. on Decision and Control*, 1988.
- [19] G. Lindstrom, J. Maluszynski, and T. Ogi, "Using Types to Interface Functional and Logic Programming," July 1990, 10 pp. technical summary submitted to 1991 SIGPLAN Symposium on Principles of Programming Languages.
- [20] A. Nerode and J. B. Remmel, "A Model for Hybrid Systems," Presented at the Hybrid Systems Workshop, Mathematical Sciences Institute, Cornell University, May 1991.
- [21] C. M. Özveren, *Analysis and Control of Discrete Event Dynamic Systems : A State Space Approach*, Ph.D. Thesis, Massachusetts Institute of Technology, August 1989.
- [22] C. M. Özveren and A. S. Willsky, *Aggregation and Multi-Level Control in Discrete Event Dynamic Systems*, Technical Report CICS-P-199, Center for Intelligent Control Systems, Massachusetts Institute of Technology, March 1990.
- [23] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *SIAM Journal of Control and Optimization*, January 1987.
- [24] P. J. Ramadge and W. M. Wonham, "Modular Feedback Logic for Discrete Event Systems", *SIAM Journal of Control and Optimization*, September 1987.

- [25] R. F. Riesenfeld, "Mathematical Methods in Computer Aided Geometric Design," chapter Design Tools for Shaping Spline, Academic Press 1989.
- [26] T. M. Sobh and R. Bajcsy, "A Model for Observing a Moving Agent," *Proceedings of the Fourth International Workshop on Intelligent Robots and Systems (IROS '91)*, Osaka, Japan, November 1991.
- [27] R. Suri, "Perturbation Analysis : The State of the Art and Research Issues Explained via the GI/G/1 Queue", *Proc. of the IEEE*, January 1989.
- [28] M. Swanson, R. Kessler, "Domains : efficient mechanisms for specifying mutual exclusion and disciplined data sharing in concurrent scheme," *First U.S./Japan Workshop on Parallel*, August 1989
- [29] J. A. Thingvold and E. Cohen, "Physical Modeling with B-Spline Surfaces for Interactive Design and Animation," *Proceedings of the 1990 Symposium on Interactive 3-D graphics*, ACM, March 1990.
- [30] A. F. Vaz and W. M. Wonham, "On Supervisor Reduction in Discrete-Event Systems", Technical Report, Systems Control Group, Department of Electrical Engineering, University of Toronto, 1985.
- [31] Y. Willner and M. Heymann, *On Supervisory Control of Concurrent Discrete-Event Systems*, Technical Report CIS-9009, Computer Science Department, TECHNION - Israel Institute of Technology, October 1990.

Appendix A

```

/* Code Generated for a Simple DRFSM */

main()
{

    /* Some initializations */

    VTV_ptr = get_VTV() ;
    drfsm (VTV_ptr) ;

    /* Finish Up */

}

/*****/

drfsm (VTV_ptr)
{
    /* do some initializations for each level */

    state_A (VTV_ptr) ;

    /* do some cleaning */
}

/*****/

state_A (VTV_ptr)
{
    int finish = 0 ;

    /* do something */

    while ( !finish ){
        get-action1 (x) ;
        if ((x > VTV_ptr[1]) && (x < VTV_ptr[2])){
            finish = 1 ;
            state_B (VTV_ptr) ;
        }
    }
}

```

```

/*****/

state_B (VTV_ptr)
{
    int finish = 0 ;

    /* do something */

    while ( !finish ){
        get-action2 (c, w) ;
        if ((c > 0.2) && (c < 0.5)){
            finish = 1 ;
            state_A (VTV_ptr) ;
        }
        if (w > 120){
            finish = 1 ;
            state_C (VTV_ptr) ;
        }
    }
}

/*****/

state_C (VTV_ptr)
{
    int finish = 0 ;

    /* do something */

    while ( !finish ){
        get-action3 (L,z,y) ;
        if (z > 3*VTV_ptr[4]/sin(VTV_ptr[5])){
            finish = 1 ;
            state_B (VTV_ptr) ;
        }
        if (y > VTV_ptr[3]){
            finish = 1 ;
            state_D (VTV_ptr) ;
        }
        if (L > 10){
            newVTV_ptr = get_VTV() ;
            drfsm (new_VTV_ptr) ;
            /* Free memory allocated to new_VTV */
            /* Complete something */
        }
    }
}

```

