A Dynamic Framework for Intelligent Inspection

Tarek M. Sobh, Chris Jaynes, and Thomas C.Henderson[1]

UUCS-92-035

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

October 15, 1992

Abstract

CAD (Computer Aided Design) typically involves the design, manufacture and inspection 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. We propose to explore the feasibility of a new approach to inspection and reverse engineering applications. In particular, we investigate the use of discrete event dynamic systems (DEDS) to guide and control the active exploration and sensing of mechanical parts for industrial inspection. The proposed framework utilizes DEDS for constructing an observer for inspection purposes.

[1]This 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.

1 Introduction

Developing environments for inspection applications is an essential activity in many engineering disciplines. Usually, too much time 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. The technique to be explored 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.

2 Objectives and 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 activities. In particular, we will 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.

Our thesis is that :

- Discrete Event Dynamical Systems (DEDS) provide the base for defining consistent and adaptive control structures for the inspection and reverse engineering problem.

If this is true, then we will be able to answer the following questions :

- What is a suitable algorithm to coordinate sensing, inspection, design and manufacturing ?
- What is a suitable control strategy for sensing the mechanical part ?
- Which parts should be implemented in hardware vs. software ?
- What are suitable language tools for constructing a reverse engineering and/or inspection strategy ?

We describe DEDS in more detail later, but they 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 [11,13].

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 will allow us to study the subdivision of the solution into reliable, reversible, and an easy-to-modify software and hardware environments.

3 Methodology for Inspection

In this section we describe the solution methodology and discuss the components separately. The control flow is also described and the methods, specific equipment and procedures to be designed and implemented 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 inspected by the co-operation of the observer camera and the probing CMM, a DEDS is used as the high-level framework for exploring the mechanical part. We next discuss DEDS in general and then we proceed to apply the framework for sensing the inspection process.

3.1 Discrete Event Dynamic Systems

Discrete event dynamic systems are dynamic systems (typically asynchronous) in which state transitions are triggered by the occurrence of discrete events in the system. 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 [2,10,11]. 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 can represent a DEDS by the following quadruple:

$$G = (X, \Sigma, U, \Gamma)$$

where X is the finite set of states, Σ is the finite set of possible events, U is the set of admissible control inputs consisting of a specified collection of subsets of Σ, corresponding to the choices of sets of controllable events that can be enabled and $\Gamma \subseteq \Sigma$ is the set of observable events.

We can visualize the concept of DEDS by means of the example in Figure 1. The graphical representation is quite similar to a classical finite automaton. Here, circles denote states, and events are represented by arcs. The first symbol in each arc label denotes the event, while the symbol following "/" denotes the corresponding output (if the event is observable). Finally, we mark the controllable events by ":u". Thus, in this example, $X = \{0, 1, 2, 3\}$, $\Sigma = \{\alpha, \beta, \delta\}$, $\Gamma = \{\alpha, \delta\}$, and δ is controllable at state 3 but not at state 1.

An *alive* state is a state that can never undergo transitions leading to a state that has no outgoing transitions (a *dead* state). A system A is *alive* if all its states are *alive*. Stability can be defined with respect to the states of a DEDS automaton. Assuming that we have identified the set of "good" states, E, that we would like our DEDS to "stay within" or to not stay outside for an infinite time, then stabilizability can be formally defined as follows:

Given a live system A and some $E \subset X$, $x \in X$ is *stabilizable* with respect to E (or E-stabilizable) if there exists a combination of controllable events (control pattern) K such that x is alive and does not stay outside E forever (E-stable) when K is used. A set of states, Q, is a *stabilizable set* if there exists a control pattern K so that every $x \in Q$ is alive and stable in A_K (A under the control pattern K), and A is a *stabilizable system* if X is a stabilizable set.

A DEDS is termed *observable* if we can use any sequence of observable events to determine the current state exactly at intermittent points in time separated by a bounded number of events. More formally, take any sufficiently long string, s, that can be generated from any initial state x. For any observable system,

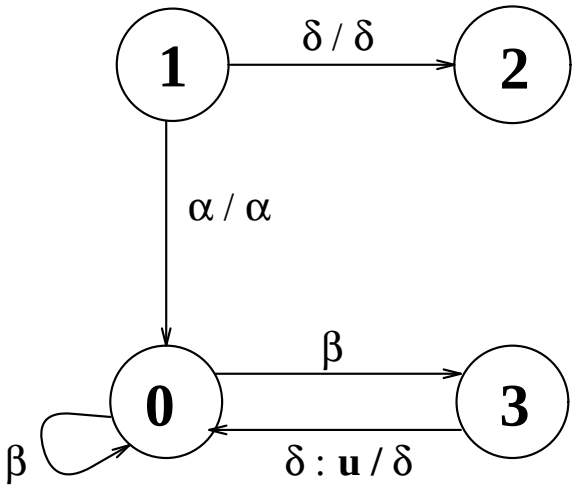


Figure 1: A Simple DEDS Example

we can then find a prefix p of s such that p takes x to a unique state y and the length of the remaining suffix is bounded by some integer n_o. Also, for any other string t, from some initial state $x^{'}$, such that t has the same output string as p, we require that t takes $x^{'}$ to the same, unique state y.

The basic idea behind strong output stabilizability is that we will know that the system is in state E iff the observer state is a subset of E. The compensator should then force the observer to a state corresponding to a subset of E at intervals of at most a finite integer i of observable transitions. If Z is the set of states of the observer, then : A is strongly output E-stabilizable if there exists a state feedback K for the observer O such that O_K is stable with respect to $E_O = \{\hat{x} \in Z \mid \hat{x} \subset E\}$.

We advocate an approach in which a stabilizable semi-autonomous visual sensing interface would be capable of making decisions about the *state* 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.

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. 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.

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 predetermined 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.

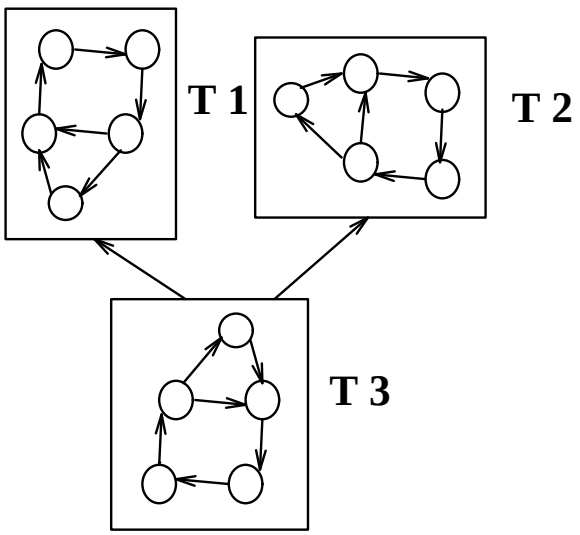


Figure 2: A Hierarchy of Tasks.

Hierarchical Representation

Figure 2 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 the observer hierarchy of models allows coarse to fine shift of attention in recovering events and asserting state transitions.

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. A graphical representation of the mapping module is shown in Figure 3. 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.

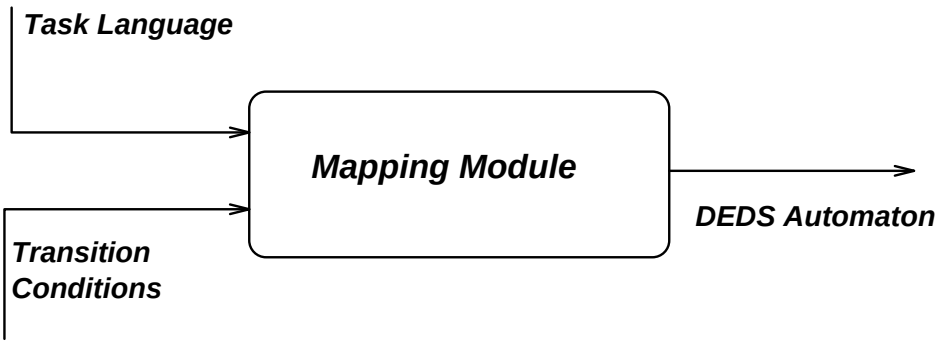


Figure 3: The Mapping Module.

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 :

$$L = hh^*h_oh_o^*h_ch_c^*$$

and a corresponding DEDS automaton is shown in Figure 4.

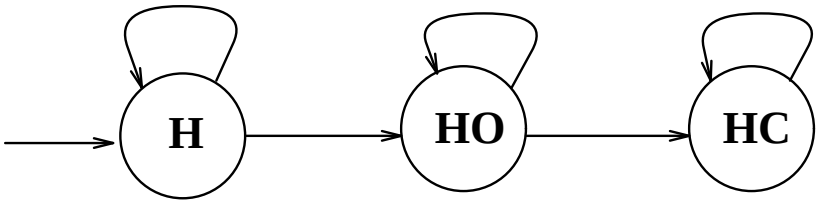


Figure 4: An Automaton for Simple Grasping.

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 mounted on a tripod, and a coordinate measuring machine (CMM) to sense the mechanical part. 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 [2,10,11]. 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 [13]. 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.

4 Experiment

An experiment was performed for inspecting a mechanical part using a camera and the coordinate measuring machine. A predefined DEDS state machine was used as the observer agent skeleton and the mechanical

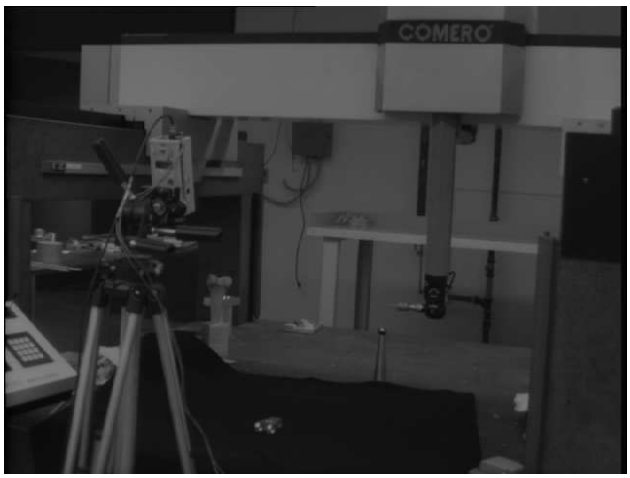


Figure 5: Experimental Setup

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 5.

The experiment proceeds as follows. First, an image is 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 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.

4.1 DEDS State Machine

The specific automata chosen for the test was a state machine G. Where $X = \{\text{Initial,EOF,Error,A,B,C,D}\}$ 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 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.
- Error: An invalid signal was received.
- EOF: The End of File signal was received.

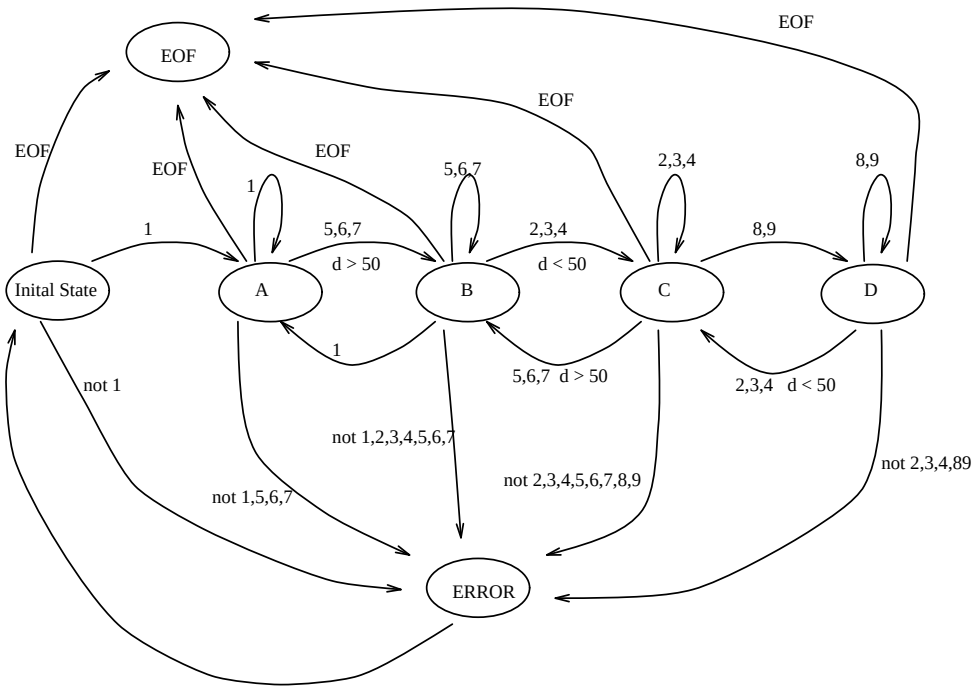


Figure 6: State Machine Used in Test

4.2 Vision and Touch

A layer of processing must exist between the state machine and the camera that provides the proper input signals. This layer is made up of several vision filters that are applied to each image as it is captured. Several things must be known about the scene before a signal is produced. The location of the part, the location of the probe, the pixel distance between them, the number of features on the part, and the distance to the closest feature.

Because the part to be inspected is known before the test is run, we are able to adjust the vision filters for the task at hand. We are also able to make specific assumptions about the images we expect to receive. The vision problem, then, becomes a simple series of filters.

First the image is thresholded at a gray-level that optimizes the loss of background while retaining the part and probe. Next, a median filter is applied that removes small regions of noise. The image is then parsed to find all segments separated by an appropriate pixel distance and labels them with a unique region identifier.

We are able to assume that the probe, if in the scene, will always intersect the image border. The probe tip is the farthest point on the probe region from the border. This holds true because of the geometry of the probe. An image with one region, that intersects the border, is the case in which the probe is touching the part.

If we have more than one region, we must discover the distance between the tip of the probe region and the part. This is done through an edge following algorithm that gives us the x, y positions of the pixels on the edge of each region. We then find the Euclidean distances between the edge points and the probe tip.

The closest point found is used in producing the signal to the state machine.

Features on the part are considered to be edge responses that are interior to the part boundary. Distance to the closest feature from the probe tip is found using the same method as before.

4.3 Results

A typical sequence from a probing task was run. The probe was introduced to 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, and error state is reached. Each image was displayed on a terminal window as it was captured along with the corresponding state of the automaton.

5 Current Developments

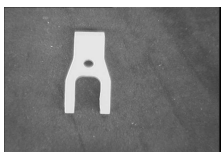
The application environment we eventually intend to develop consists of three major working elements: the sensing, design, and manufacturing modules. The ultimate goal is to establish a computational framework that is capable of deriving designs for machine parts or objects, inspect and refine them, 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, some other parts of the system may be possible to put in hardware, but 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.

5.1 Robotics and Sensing

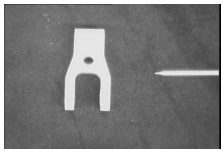
We intend to use a robot arm (a PUMA 560), a vision sensor (B/W CCD camera) mounted on the end effector and a coordinate measuring machine (CMM). A discrete event dynamic system (DEDS) algorithm will be used to coordinate the movement of the robot sensor and the CMM. The DEDS control algorithm will also guide the CMM to the relevant parts of the objects that need to be explored in more detail (curves, holes, complex structures, etc.)

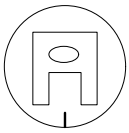
5.2 Computer Aided Design and Manufacturing

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 [12,15] 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

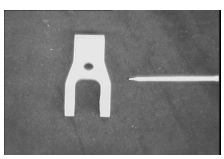


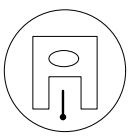
STATE A: Probe Not In Scene



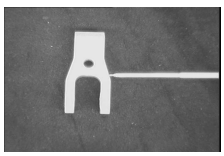


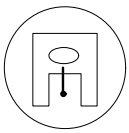
STATE B: Probe Approaching Object



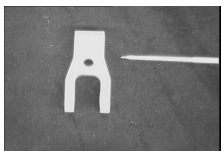


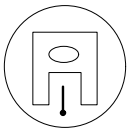
STATE C: Probe Close To Object



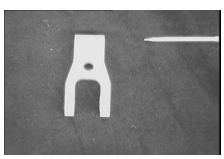


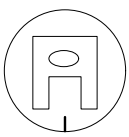
STATE D: Probe Touching Object



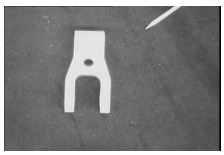


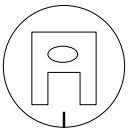
STATE C: Probe Close To Object



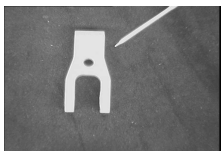


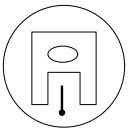
STATE B: Probe Approaching Object



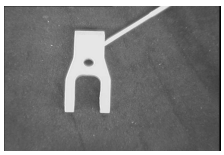


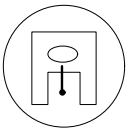
STATE B: Probe Approaching Object



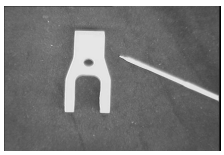


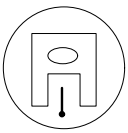
STATE C: Probe Close To Object



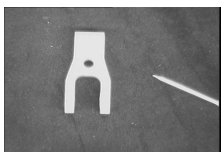


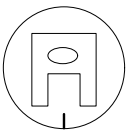
STATE D: Probe Touching Object



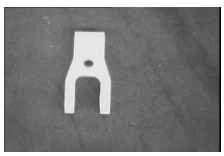


STATE C: Probe Close To Object

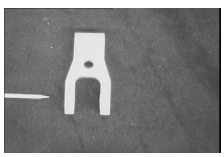


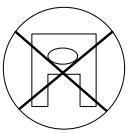


STATE B: Probe Approaching Object



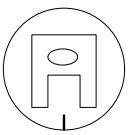
STATE A: Probe Not In Scene





ERROR STATE

STATE A: Probe Not In Scene



STATE B: Probe Approaching Object

STATE A: Probe Not In Scene

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 [6]. 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.

5.3 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 [4,7] and one of the sub-problems would be to study the need and efficiency of making customized chips in the environment. The DEDS 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 DEDS 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 [9,14]. We also intend to use a graphical behavior design tool [3] for the automatic production of the sensing DEDS automata code, from a given control language description.

6 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 control strategies for sensing the world and coordinating the different activities between the phases. We use a DEDS framework to construct an intelligent observer module for inspection. 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] 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.

[3] M. J. Bradakis, "Reactive Behavior Design Tool," Master's Thesis, Computer Science Department, University of Utah, January 1992.

[4] 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.

[5] 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.

[6] S. Drake and S. Sela, "A Foundation for Features," *Mechanical Engineering*, 111(1), January 1989.

[7] J. Gu and K. Smith, "A Structured Approach for VLSI Circuit Design," *IEEE Computer*, 22(11), 1989.

[8] 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.

[9] 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.

[10] A. Nerode and J. B. Remmel, "A Model for Hybrid Systems," Presented at the Hybrid Systems Workshop, Mathematical Sciences Institute, Cornell University, May 1991.

[11] C. M. Özveren, *Analysis and Control of Discrete Event Dynamic Systems : A State Space Approach*, Ph.D. Thesis, Massachusetts Institute of Technology, August 1989.

[12] R. F. Riesenfeld, "Mathematical Methods in Computer Aided Geometric Design," chapter Design Tools for Shaping Spline, Academic Press 1989.

[13] 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.

[14] 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.

[15] 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.