Autonomous Observation

Tarek M. Sobh[1]

UUCS-92-041

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

December 23, 1992

Abstract

We address the problem of observing an agent. We advocate a modeling approach for the visual system and its observer, where a discrete event dynamic system (DEDS) framework is developed and "events" are defined as ranges on parameter subsets. The dynamic recursive context for finite state machines (DRFSM) is described with some applications in the inspection and reverse engineering domain. We propose a system for observing a manipulation process, where a robot hand manipulates an object. We recognize the hand/object interaction over time and a stabilizing observer is constructed. Low-level modules are developed for recognizing the events that causes state transitions within the dynamic manipulation system. The work examines closely the possibilities for errors, mistakes and uncertainties in the manipulation system, observer construction process and event identification mechanisms. The DRFSM DEDS systems utilizes different tracking techniques in order to observe and recognize tasks and agents in an *active*, *adaptive* and *goal-directed* manner.

[1]This work was supported in part by DARPA grant N00014-91-J-4123 and NSF grant CDA 9024721; Air Force AFOSR Grants 88-0244, 88-0296; Army/DAAL Grant 03-89-C-0031PRI; NSF Grants CISE/CDA 88-22719, IRI 89-06770; DARPA Grant N0014-88-0630 and DuPont Corporation. 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

The problem of observing an agent was addressed in the literature extensively. It was discussed in the work addressing tracking of targets and, determination of the optic flow [3,12,13,26,29,54], recovering 3-D parameters of different kinds of surfaces [11,19,21,35,43,44,51,52], and also in the context of other problems [2,6,7,10,48].

Recovering the visual parameters of a scene under observation and using them to develop methods for tracking moving agents within a dynamic scene was discussed [13,14,15,22,23,41,54]. However, the need to *recognize*, *understand* and *report* on different visual steps within a dynamic task was not sufficiently addressed. In particular, there is a need for high-level symbolic interpretations of the actions of an agent that attaches meaning to the 3-D world events, as opposed to simple recovery of 3-D parameters and the consequent tracking movements to compensate their variation over time. Thus, the need arises for some kind of an *intelligent observer* to understand the actions of a dynamic agent.

In this work we establish a framework for the general problem of observation, recognition and understanding of dynamic visual systems, which may be applied to different kinds of visual tasks. We establish "intelligent" high-level control mechanisms for the observer in order to achieve an efficient approach to visually recognizing different processes within a dynamic system. The observation process can be thought of as a stage in a closed-loop fully or semi-automated system where there are robots who perform the required task and some other robots who observe them and correct their actions when something goes wrong.

1.1 General Framework

To be able to observe what an agent looks like and/or how it moves, we must be able to identify how the it moves and how does the agent/world physical relationship evolves over time. An obvious way of doing this would be to identify the motion vectors as seen be the observer. In other words, identify the two-dimensional vectors in the observer's camera plane and use these as a cue to know how the objects under consideration moves in the three-dimensional space. The problems of recovering the image flow vectors (the two-dimensional motion vectors in the camera plane), and identifying the scene structure and motion have been key problems in computer vision. Many techniques have been developed for estimating the image flow [3,12,20,26,29], and to recover the three-dimensional world structure and motion [9,19,44,46,47,50,52]. Those techniques are not problem-oriented, they are not restricted to a particular problem domain, as is the case with the observer construction problem.

Using the above techniques directly to solve the observer problem will not be efficient. In fact, possibly not feasible to perform in a practical way using the current technology, as the complexity of the world increases. Due to the fact that we probably know a-priori some information about the allowable (or useful) processes and the geometry of the agent, posing the problem as a structure-from-motion vision procedure is a very naive way of modeling the observer system. It should also be noted that the observer will have to be an *active* one to be able to interact with the environment in such a way as to be able to "see" at all times. The idea of an active observer was discussed in the literature [2,6], and it was shown that an active observer can solve basic vision problems in a much more efficient way than a passive one.

We use a discrete event dynamic system as a high-level structuring technique to model the visual system. Our formulation uses the knowledge about the system and the different actions in order to solve the observer problem in an efficient, stable and practical way. The model incorporates different relationships and the possible errors in the actions. It also uses different tracking mechanisms so that the observer can keep track of the workspace of the changing environment. Low-level modules are developed for recognizing the "events" that causes state transitions within the dynamic system. The process uses a coarse quantization of the actions in order to attain an active, adaptive and goal-directed sensing mechanism.

1.2 Visual Uncertainties

The work examines closely the possibilities for errors, mistakes and uncertainties in the visual, observer construction process and event identification mechanisms. We divide the problem into six major *levels* for developing uncertainty models in the observation process. The *sensor* level models deals with the problems in mapping 3-D features to pixel coordinates and the errors incurred in that process. We identify these uncertainties and suggest a framework for modeling them. The next level is the *extraction strategy* level, in which we develop models for the possibility of errors in the low-level image processing modules used for identifying features that are to be used in computing the 2-D evolution of the scene under consideration and computing the image flow . In the third level, we utilize the geometric and mechanical properties of the agent and/or objects to reject unrealistic estimates for 2-D movements that might have been obtained from the first two levels.

After having obtained 2-D models for the evolution of the relationship, we transform the 2-D uncertainty models into 3-D uncertainty models for the structure and motion of the entire scene. The fourth level uses the equations that govern the 2-D to 3-D relationship to perform the conversion. The fifth level rejects the improbable 3-D uncertainty models for motion and structure estimates by using the existing information about the geometric and mechanical properties of the moving components in the scene. The sixth and highest level is the DEDS formulation with uncertainties, in which state transitions and event identification is asserted according to the 3-D models of uncertainty that were developed in the previous levels.

We describe the automaton model of a discrete event dynamic system (DEDS) in the next section and then proceed to formulate our framework for the observer construction. The Dynamic Recursive Context for Finite State Machines (DRFSM) is then described, with some applications. Then we develop efficient low-level event-identification mechanisms for determining different movements in the system and for moving the observer. Next, the uncertainty levels are described in details.

2 Discrete Event Dynamic Systems

Discrete event dynamic systems (DEDS) 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 [24,34,36,38,39]. 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.

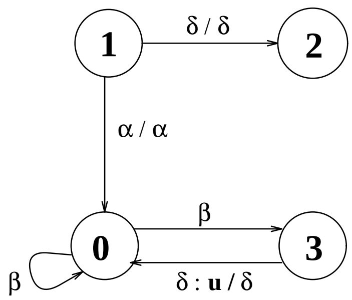


Figure 1: A Simple DEDS Example

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, 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 agent. Thus providing both symbolic and parametric descriptions to the control module. The DEDS-based active sensing interface will be discussed in the following section.

3 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 an observation or exploration process 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 a touching probe and a mechanical 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.

As an example, robotic manipulation actions can be modeled efficiently within a discrete event dynamic system framework. It should be noted that we do not intend to *discretize* the workspace of the manipulating robot hand or the movement of the hand, we are merely using the DEDS model as a high level structuring technique to preserve and make use of the information we know about the way in which each manipulation task should be performed, in addition to the knowledge about the physical limitations of both the observer and manipulating robots. We avoid the excessive use of decision structures and exhaustive searches when observing the 3-D world motion and structure.

A bare-bone approach to solving the observation problem would have been to visually reconstruct the full 3-D motion parameters of the robot's hand, which would have more than six degrees of freedom, depending on the number of fingers and/or claws and how they move. The object's motion should also be recovered in 3-D, which is complicated, especially if it is a non-rigid body. That process should be done in real time while the task is being performed. A simple way of tracking is to keep a fixed geometric relationship between the observer camera and the hand over time. However, the above formulation is inefficient, not needed and for all practical purposes infeasible to compute in real time. The limitation of the observer reachability and the extensive computations required to perform the visual processing are motives behind formulating the problem as a hierarchy of task-oriented observation modules that exploits the higher-level knowledge about the existing system, in order to achieve a feasible mechanism of keeping the visual process under supervision.

We do a coarse quantization of the visual manipulation actions which has both continuous and discrete aspects of manipulation dynamics. State transitions within the manipulation domain are asserted according to probabilistic models that determine at different instances of time whether the visual scene under inspection has changed its state within the discrete event dynamic system state space. We next discuss building the manipulation model for two simple tasks, grasping and screwing, then we proceed to develop the observer for these tasks. Formulating the uncertainty models for the state transitions and the inter-state continuous dynamics will be left for the sections that deal with the different uncertainty levels and event identification mechanisms.

3.1 Building the Model

The ultimate goal of the observation mechanism is to be able to know at all (or most) of the time what is the current manipulation process and what is the visual relationship between the hand and the object. It should be noticed that this concept is very similar to the concept of *observability* as defined in the previous section for general DEDS. The fact that the observer will have to move in order to keep track of the manipulation process, makes one think of the output feedback stabilizability principle for general DEDS as a model for the tracking technique that has to be performed by the observer's camera.

In real-world applications, many manipulation tasks are performed by robots, including, but not limited to, lifting, pushing, pulling, grasping, squeezing, screwing and unscrewing of machine parts. Modeling all the possible tasks and also the possible order in which they are to performed is possible to do within a DEDS state model. The different hand/object visual relationships for different tasks can be modeled as the set of states X. Movements of the hand and object, either as 2-D or 3-D motion vectors, and the positions of the hand within the image frame of the observer's camera can be thought of as the events' set Γ that causes state transitions within the manipulation process. Assuming, for the time being, that we have no direct control over the manipulation process itself, we can define the set of admissible control inputs U as the possible tracking actions that can be performed by the hand holding the camera, which actually can alter the visual configuration of the manipulation process (with respect to the observer's camera). Further, we can define a set of "good" states, where the visual configuration of the manipulation process enables the camera to keep track and to know the movements in the system. Thus, it can be seen that the problem of observing the robot reduces to the problem of forming an output stabilizing observer for the system under consideration, which was discussed in details in the previous section.

It should be noted that a DEDS representation for a manipulation task is by no means unique, in fact, the degree of efficiency depends on the person who builds the model for the task, testing the optimality of a manipulation models is an issue that is to be addressed in the future. Automating the process of building a model is another issue that will have to be addressed later. As the observer identifies the current state of a manipulation task in a non ambiguous manner, it can then start using a practical and efficient way to determine the next state within a predefined set, and consequently perform necessary tracking actions to stabilize the observation process with respect to the set of good states. That is, the current state of the system tells the observer what to *look for* in the next step.

3.1.1 A Grasping Task

We present a simple model for a grasping task. The model is that of a gripper approaching an object and grasping it. The task domain was chosen for simplifying the idea of building a model for a manipulation task. It is obvious that more complicated models for grasping or other tasks can be built. The example shown here is for illustration purposes.

As shown in Figure 2, the model represents a view of the hand at state 1, with no object in sight, at state 2, the object starts to appear, at state 3, the object is in the claws of the gripper and at state 4, the claws of the gripper close on the object. The view as presented in the figure is a frontal view with respect to the camera image plane, however, the hand can assume any 3-D orientation as so long as the claws of the gripper are within sight of the observer, for example, in the case of grasping an object resting on a tilted planar surface. This demonstrates the continuous dynamics aspects of the system. In other words, different orientations for the approaching hand are allowable and observable. State changes occur only when the object appear in sight or when the hand encloses it. The frontal upright view is used to facilitate drawing the automaton only. It should be noted that these states can be considered as the set of good states E, since these states are the expected different visual configurations of a hand and object within a grasping task.

States 5 and 6 represent instability in the system as they describe the situation where the hand is not centered with respect to the camera imaging plane, in other words, the hand and/or object are not in a good visual position with respect to the observer as they tend to escape the camera view. These states are considered as "bad" states as the system will go into a non-visual state unless we correct the viewing position. The set $X = \{1, 2, 3, 4, 5, 6\}$ is the finite set of states, the set $E = \{1, 2, 3, 4\}$ is the set of "good" states.

The events are defined as motion vectors or motion vector probability distributions, as will be described later, that causes state transitions and as the appearance of the object into the viewed scene. The transition from state 1 to state 2 is caused by the appearance of the object. The transition from state 2 to state 3 is caused by the event that the hand has enclosed the object, while the transition from state 3 to state 4 is caused by the inward movement of the gripper claws. The transition from the set $\{1, 2\}$ to the set $\{5, 6\}$ is caused by movement of the hand as it escapes the camera view or by the increase in depth between the camera and the viewed scene, that is, the hand moving far away from the camera. The self loops are caused by either the stationarity of the scene with respect to the viewer or by the continuous movement of the hand as it changes orientation but without tending to escape a good viewing position of the observer. In the next section we discus different techniques to identify the events. The controllable events denoted by "$: t$" are the tracking actions required by the hand holding the camera to compensate for the observed motion. Tracking techniques will later be addressed in detail. All the events in this automaton are observable and thus the system can be represented by the triple $G = (X, \Sigma, T)$, where X is the finite set of states, Σ is the finite set of possible events and T is the set of admissible tracking actions or controllable events.

It should be mentioned that this model of a grasping task could be extended to allow for error detection and recovery. Also search states could be added in order to "look" for the hand if it is no where in sight. The purpose of constructing the system is to develop an observer for the automaton which will enable us

Figure 2: A Model for a Grasping Task

to determine the current state of the system at intermittent points in time and further more, enable us to use the sequence of events and control to "guide" the observer into the set of good states E and thus stabilize the observation process. Disabling the tracking events will obviously make the system neither stable or pre-stable with respect to the set $E = \{1, 2, 3, 4\}$, however, it should be noted that the subset $\{3, 4\}$ is already stable with respect to E regardless of the tracking actions, that is, once the system is in state 3 or 4, it will remain in E (as defined by our formulation of the model). The whole system is stabilizable w.r.t. E, enabling the tracking events will cause all the paths from any state to go through E in a finite number of transitions and then will visit E infinitely often.

3.1.2 A Screwing Task

The next model we present is one for a simple screwing task. The task is that of a gripper screwing an object (a nail for example). It is assumed that the claws of the gripper already encloses the nail and that contact is maintained throughout the process, the rotation is allowed to be either clockwise or anticlockwise.

As shown in Figure 3, the model represents a frontal view of the hand at state 1, with the object between the claws, the hand starts to rotate at state 2 and 3 with some view of the claws and the object still in sight and the claws are occluded at state 4 which represents a side view of the gripper. This specific visual representation was chosen because of the fact that transitions between states 1 and 3 and the self loop at 3 cannot be compensated by a tracking action due to the physical limitations of the tracking arm,

Figure 3: A Model for a Screwing Task

in other words, the observing robot might not be able to do 360 degrees rotations around the manipulating hand, especially if the workspaces of both robots do not intersect and both are fixed, non-mobile robots. As mentioned before, the frontal upright view with respect to the camera imaging plane in state one was chosen only to facilitate drawing the automaton. The hand can assume any 3-D orientation as so long as the claws in states 1, 2 and 3 are within sight of the observer, for example, in the case of screwing a nail into a tilted wall.

As shown by our model, the automaton tends to keep the frontal view of the hand as long as possible (as far as the observer robot can rotate), after that the observer will just have to sit idle until rotation of the hand is trackable again. If one define the stable visual state as state 1, then obviously the system cannot be made stable with respect to that state, however, one can think of a screwing action on the whole as a stable set, since the robot hand is always within sight of the observer and it does not tend to escape the viewing field. In that case the set of "good" states E is the same as the set $X = \{1, 2, 3, 4\}$, the finite set of states. The goal of the observer in that case would basically be trying to keep a frontal view as long as it can.

The event e_1 can be defined as rotations that the observer robot can track and keep a frontal position of the hand, while e_2 is the one that makes the observable robot reach its "limit" position where it cannot rotate around the hand in the same direction any longer. The rotations e_3 are the untrackable rotations, which lie beyond the reachable workspace of the observable robot. The event e_4 can be defined as the event that causes the visual scene to be a side view of the gripper.

3.2 Developing the Observer

In order to know the current state of the manipulation 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. An observer, as defined in the previous section, have to be constructed according to the visual system for which we developed a DEDS model. 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 that the user may specify depending on the nature of the process. It should be noticed that events can be asserted with a specific probability as will be described in the sections to come and thus state transitions can be made according to pre-specified thresholds that compliments each state definition. In the case of developing ambiguities in determining current and future states, the history of evolution of past event probabilities can be used to navigate backwards in the observer automaton till a strong match is perceived, a fail state is reached or the initial ambiguity is asserted.

As an example, for the model of the grasping task, an observer can be formed for the system as shown in Figure 4. It can be easily seen that the system can be made stable with respect to the set E_O as defined in the previous section.

At the beginning, the state of the system is totally ambiguous, however, the observer can be "guided" to the set E_O consisting of all the subsets of the good states E as defined on the visual system model. It can be seen that by enabling the tracking event from the state (5, 6) to the state (1, 2), all the system can be made stable with respect to E_O and thus the system is strongly output stabilizable. The singleton states represent the instances in time where the observer will be able to determine without ambiguity the current state of the system.

In the next sections we shall elaborate on defining the different events in the visual manipulation system and discuss different techniques for event and state identification. We shall also introduce a framework for computing the uncertainty in determining the observable visual events in the system and a method by which the uncertainty distribution in the system can be used to efficiently keep track of the different observer states and to navigate in the observer automaton.

3.3 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 4: Observer for the Grasping System

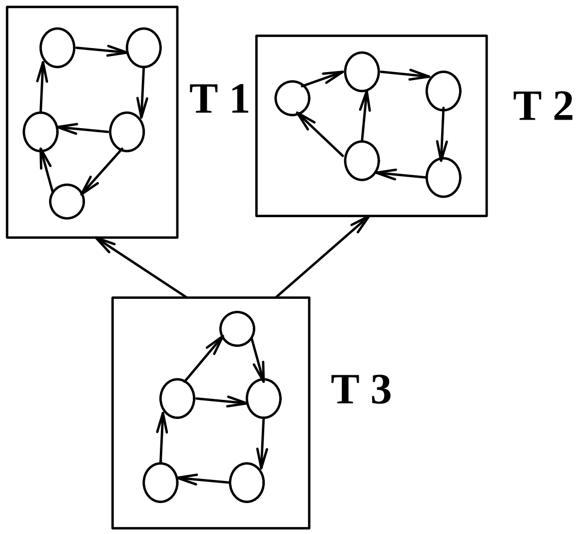


Figure 5: A Hierarchy of Tasks.

3.4 Hierarchical Representation

Figure 5 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.

3.5 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 6.

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

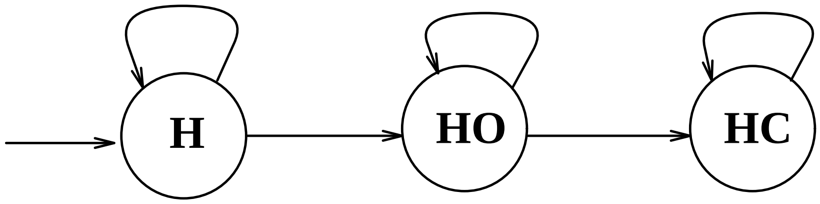


Figure 6: An Automaton for Simple Grasping.

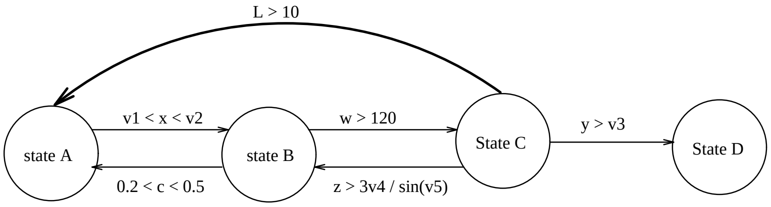
strings that the operator supplies do not represent the task language correctly, and in that case some learning techniques would have to be implemented which, in effect, augment the input set of strings into a language that satisfies some pre-determined criteria. For example, y^* is substituted for any 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.

4 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 existing reactive behavior design tools that are 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.

4.0.1 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 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.



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

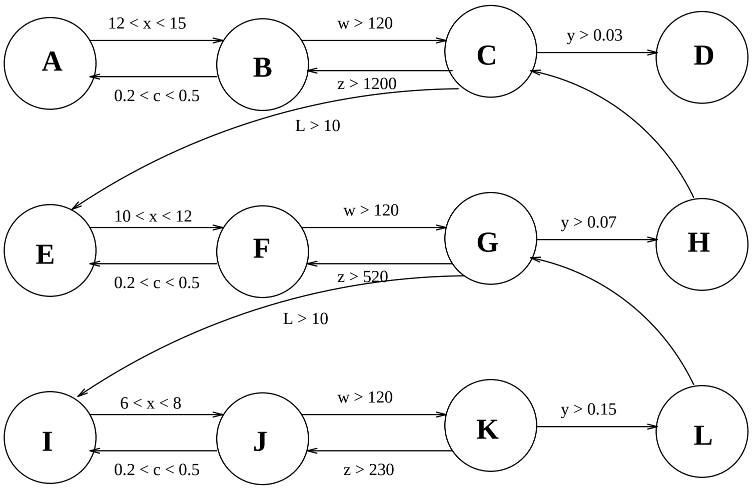


Figure 8: Flat Representation of a Simple DRFSM

4.0.2 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 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. 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. In many cases, however, it is impossible to build the equivalent FSM for a process because some values of its

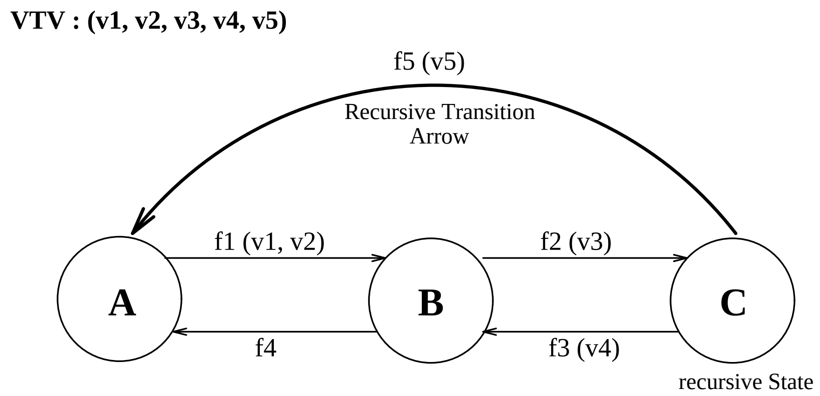


Figure 9: New Notation

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.

4.0.3 Implementation of DRFSM

We intend to develop extensions to reactive behavior design tools 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 9.

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.
- For each RTA in the figure build a user-defined function: Get_New_VTV to be filled by the user of the reactive behavior design tool 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.

4.0.4 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.
- DRFSM Design: Use the reactive behavior design tool to draw the DRFSM and generate the corresponding C code.
- Implementation: The code generated by the reactive behavior design tool 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.

4.0.5 Applying DRFSM in Feature extraction

As an example, We use a B/W CCD camera and a coordinate measuring machine (CMM) to sense a 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. The DRFSM 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. 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.

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 10.

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 >

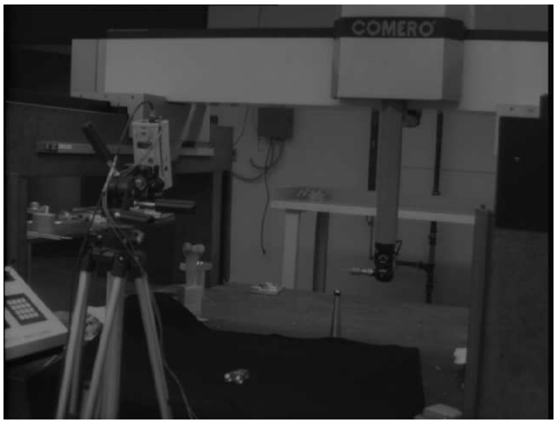


Figure 10: Experimental Setup

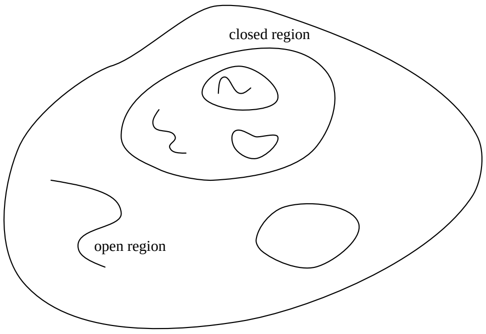


Figure 11: An Example for a Recursive Object

< term > :: O | < feature >

For example, the symbolic notation of Figure 11 is

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

Figure 12 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 13 shows a simple DRFSM DEDS machine for the exploration and inspection of mechanical parts, using both active vision and touch sensors.

5 Event Identification

In this section we discuss different techniques for calculating the "events" that causes state transitions within the model that we discussed in the previous sections. We introduce the concept of uncertainty in recovering the visual actions of the manipulation process, as an example, and formulate a way of using the uncertainty in the system in an efficient recovery mechanism. Using the formulation in the previous section, it can be shown, from the examples used in modeling the manipulation process, that the events that causes state transitions are either primitives like specific 3-D movements of the manipulating hand and/or events like "there is an object now in view", "the hand has enclosed the object" and so on. The

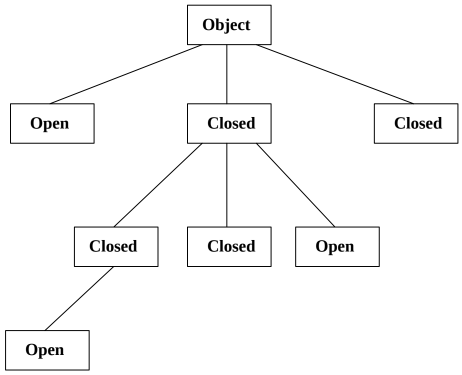


Figure 12: Graph for the Recursive Object

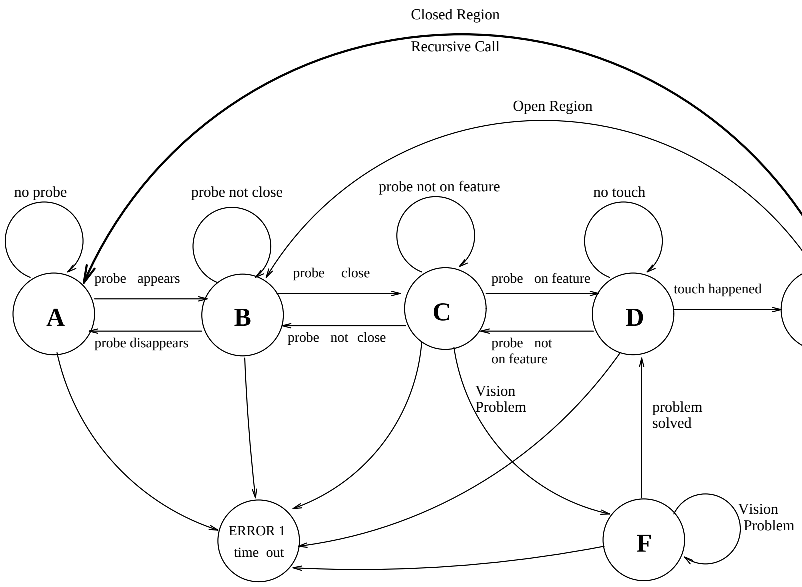


Figure 13: A DRFSM DEDS for Inspection

events that are supposed to be identified and recovered at different states of the observer automaton are highly dependent on the *current* state in the observation process. Thus the observer tends to “look” at specific actions at different instances of time.

We next discuss techniques to be used in identifying the 3-D motion of the manipulation hand and/or the object, which are events that are always important to recover in order to enable the observer to navigate in the automaton. The process is started by identifying the manipulating hand and the object (if it exists) within the observer's viewing window. We then proceed to develop an algorithm for detecting the two-dimensional motion vectors of the hand on the observer's camera plane. Overall motion estimation and different tracking strategies are then developed in order to be able to stabilize the observer in the most efficient way.

In order to identify the manipulating hand movement within a grasping task, we use the image motion to estimate the motion. This task can be accomplished by either feature tracking or by computing the full optic flow. Feature tracking seems to be a good option for determining the hand motion, especially since the same hand will probably be used throughout the manipulation process, and if the system is to be ported to another manufacturing environment, then the interface that tracks specific features can be changed while maintaining modularity. On the other hand, determining the full optic flow seems to be essential for computing the object motion, as we might not know in advance any shape or material information about the objects to be manipulated.

Many techniques were developed to estimate the optic flow (the 2-D image motion vectors) [3,12,20,26,29,54], we propose an algorithm for calculating the image flow and then we discuss a simpler version of the same algorithm for real time detection of the 2-D motion vectors. The image flow detection technique we use is based on the sum-of-squared-differences optic flow. We consider two images, 1 and 2 as shown in Figure 14. For every pixel (x, y) in image 1 we consider a pixel area N surrounding it and search a neighboring area S to seek a corresponding area in image 2 such that the sum of squared differences in the pixel gray levels is minimal as follows :

$$SSD(\acute{x}, \acute{y}) = \min_{\acute{x},\acute{y} \in S} \sum_{\Delta x, \Delta y \in N} [E(x + \Delta x, y + \Delta y) - \acute{E}(\acute{x} + \Delta x, \acute{y} + \Delta y)]^2$$

The image flow vector of pixel (x, y) then points from the center of N in the first image to the center of the best match in the second image. The search area S should be restricted for practicality measures. In the case of multiple best matches, we can use the one which implies minimum motion, as a heuristic favoring small movements. It should be noted that the accuracy of direction and magnitude of the optic flow determination depends on the sizes of the neighborhoods N and S.

There are three basic problems with this simple approach, one is that the sum of squared differences will be near zero for all directions wherever the graylevel is relatively uniform, the second is that it suffers from the so-called “aperture problem” even if there is a significant graylevel variation. To illustrate this point, consider a vertical edge moving to the right by one pixel distance, and suppose the N window size is 3×3 pixels and the S window size is 5×5 pixels, the squared-differences at an edge point reaches its maximum for three directions as indicated by the vectors (in pixel displacements); $(1, 0)$, $(1, -1)$ and $(1, 1)$. Figure 15 illustrates the aperture problem, where the direction of motion of edge E cannot be determined by viewing E through the aperture A. The third problem is that the scheme will only determine the displacement to pixel accuracy.

Figure 14: Identifying the SSD Optic Flow

Figure 15: The Aperture Problem and Normal Flow Estimation

We solve the first problem by estimating the motion only at the hand or object pixels (as determined by the two-dimensional segmentation scheme) where the intensity changes significantly. The Sobel edge detector is applied to the first image to estimate the edge magnitude $M(x, y)$ and direction $D(x, y)$ for every pixel :

$$M(x, y) \approx \sqrt{E_x^2 + E_y^2}$$

$$D(x, y) \approx tan^{-1}\left(\frac{E_x}{E_y}\right)$$

where E_x and E_y are the partial derivatives of the first image with respect to x and y, respectively. The edge direction and magnitude is discretized depending on the size of the windows N and S. The motion is then estimated at only the pixels where the gradient magnitude exceeds the input threshold value.

Figure 16: Subpixel Accuracy for the Optic Flow

Motion ambiguity due to the aperture problem can be solved by estimating only the *normal* flow vector. It is well known that the motion along the direction of intensity gradient only can be recovered. Then we evaluate the SSD functions at only those locations that lie on the gradient directions and choose the one corresponding to the minimal SSD, if more than one minimal SSD exist we can choose the one corresponding to the smallest movement, as described above. The full flow vector can then be estimated by using the following equation which relates the normal flow vector $\vec{v}_n$ to the full flow vector $\vec{v}$.

$$\vec{v}_n = \vec{v}.\vec{n}$$

This method works under the assumption that the hand image motion is locally constant. Solving the over-determined linear system will result in a solution for the full flow. The least square error of the system can help us to decide whether the assumption is a reasonably valid one for determining the event that caused the transition in the DEDS. On the other hand, full flow determination can be performed for small clusters of points in the image and a number of full flow estimates is then used for 3-D recovery.

To obtain sub-pixel accuracy, we can fit a one-dimensional curve along the direction of the gradient for all the SSD values obtained. A polynomial of the degree of the number of points used along the gradient can be used to obtain the best precision. However, for an S window of size 7×7 pixels or less and an N window of size 3×3 or so, a quadratic function can be used for efficiency and to avoid optimizational instabilities for higher order polynomials. Subpixel accuracy using a quadratic function is shown in Figure 16. The subpixel optimum can be obtained by finding the minimum of the function used and using the displacement at which it occurred as the image flow estimate. To avoid probable discontinuities in the SSD values, the image could be smoothed first using a gaussian with a small variance.

A simpler version of the above algorithm can be implemented in real-time using a multi-resolution approach [54]. We restrict the window size of N to 3×3 and that of S to 5×5, and perform the algorithm on different levels of the gaussian image pyramid. A gaussian pyramid is constructed by the successive applications of gaussian low-pass filtering and decimation by half. The pyramid processor, PVM-1 is capable of producing complete gaussian pyramid from a 256 by 256 image in one video frame ($\frac{1}{30}$ of a second). Maxvideo boards can be used for the simultaneous estimation of image flow at all the levels of the pyramid for all the pixels. Image flow of 1 pixel at the second level would correspond to 2 pixels in the original image, 1 pixel displacement at the third level would correspond to 4 pixels in the original image, and so on. The level with the smallest least square fitting error of the normal flow can be chosen to get the full flow and the motion vector is scaled accordingly. This method is crude in the sense that it only allow image flow values of 1,2,4 or 8 pixel displacement at each pixel, but it can be used for detecting fast movements of the hand.

By either using a flow recovery algorithm or a feature identification and tracking algorithm, we end up having a set of values for 2-D displacements of a number of pixels. The problem now is to model the uncertainty in those 2-D estimates, which are to be used later for 3-D parameter recovery. For example, if the estimate is - for a specific 3-D feature - that pixel (x_i, y_j) has moved to pixel (x_m, y_n), then the problem reduces to finding space probability distributions for the four indices. The sensor acquisition procedure (grabbing images) and uncertainty in image processing mechanisms for determining features are factors that should be taken into consideration when the uncertainty in the optic flow is computed. In sections 5, 6 and 7 we discuss these problems in details.

5.1 Recovering 3-D events

One can model an arbitrary 3-D motion in terms of stationary-scene/moving-viewer as shown in Figure 17. The optic flow at the image plane can be related to the 3-D world as indicated by the following pair of equations for each point (x, y) in the image plane [35] :

$$v_x = \left\{ x\frac{V_Z}{Z} - \frac{V_X}{Z} \right\} + \left[xy\Omega_X - \left(1 + x^2\right)\Omega_Y + y\Omega_Z \right]$$

$$v_y = \left\{ y\frac{V_Z}{Z} - \frac{V_Y}{Z} \right\} + \left[\left(1 + y^2\right)\Omega_X - xy\Omega_Y - x\Omega_Z \right]$$

where v_x and v_y are the image velocity at image location (x, y), (V_X, V_Y, V_Z) and $(\Omega_X, \Omega_Y, \Omega_Z)$ are the translational and rotational velocity vectors of the observer, and Z is the unknown distance from the camera to the object.

In this system of equations, the only knowns are the 2-D vectors v_x and v_y, if we use the formulation with uncertainty then basically the 2-D vectors are random variables with a known probability distribution. In case that the real 3-D relationships between feature points (on the hand) are known, then recovering the absolute depth is a simple process, The equations can then be be formalized, in case that that the 3-D features lie on a planar surface, as follows :

$$v_x = (1 - px - qy)\left(x\frac{V_Z}{Z_o} - \frac{V_X}{Z_o} \right) + \left[xy\Omega_X - \left(1 + x^2\right)\Omega_Y + y\Omega_Z \right]$$

$$v_y = (1 - px - qy)\left(y\frac{V_Z}{Z_o} - \frac{V_Y}{Z_o} \right) + \left[\left(1 + y^2\right)\Omega_X - xy\Omega_Y - x\Omega_Z \right]$$

where Z_o is the absolute depth, p and q are the planar surface orientations. It should be noticed that the resulting system of equations is nonlinear, however, it has some linear properties. The rotational part, for example, is totally linear. In section 8 we discuss different methods for solving the system of equations and thus recovering the 3-D parameters in real time with and without uncertainty formulation.

Figure 17: Formulation for Stationary Scene/Moving Viewer

A part of the events definition, as mentioned before, is the recognition of the existence of an object, for example. In other words, identifying objects in the visual scene and not only recovering 3-D motion. Orientation of the object relative to the observer's camera and its shape can always be asserted by a simple 2-D segmentation strategy. A data base of different shapes and orientations for different sized objects with the associated state that they may be manipulated in may be used and updated by the system. Correlation-based matching techniques can be used to compare 2-D object representations, while moment computations are used to scale, shift and re-orient the shapes to be correlated. New objects can still be recognized and stored in this data base to facilitate future accesses.

5.2 The Controllable Events

The only kind of control inputs that can be supplied to the observer robot are the tracking actions. Depending on the nature of the manipulation process, the observer has to keep track of the hand and object within the camera image plane in such a way so as to be able to observe the process. The intelligent tracking control is supplied by the DEDS formulation. Simple-minded tracking ideas, like keeping fixed 3-D relation between the camera and the manipulating agent are not to be used in our system. The manipulation action might be a simple one that does not require complex tracking, such as screwing and unscrewing, however, more complex events, where the hand may occlude the manipulation process, or when the hand starts moving away from the observer, might suggest the need for complex tracking mechanisms, including translations and rotations of the observing robot hand on which the camera is mounted.

A subset of the three-dimensional motion and structure parameters would have to be calculated using two or more frames [19,47,50,52]. The size of the subset will depend on the *expected* kind of 3-D motion, as the current state of the DEDS system will specify. Our system needs to track the object while using all the six degrees of freedom of the observer robot in order to position the observer at the best feasible position at different states of the automaton. Using rotations only to follow the end effector of the manipulating robot is not sufficient for the stabilizing observer.

Two kinds of tracking mechanisms can be used, in the first kind, the two images on which the motion estimation algorithms will be used, will be taken while the camera is stationary and then the camera will move and the process will be repeated after the camera stops. The observer movement will be a "jerky" one. Another scheme can be used where the camera can grab images while the robot arm holding it is moving, in this case one should compensate for the moving arm before calculating the image flow of the hand and/or object. Thus, the problem reduces to finding the image flow due to the camera movement using the stationary-scene/moving-viewer 3-D formulation. In the absence of translations, for example, we can compensate for the rotational part in a very fast and efficient way. Compensation will have to be performed before using the structure and motion recovery algorithms.

6 Sensor Uncertainties

In this section and the next two sections we develop and discuss modeling the uncertainties in the recovered 2-D displacement vectors. There are many sources of errors and ways to model uncertainties in image processing [53]. As mentioned in the section describing techniques for recovering the image flow, the uncertainty in the recovered values results from sensor uncertainties and noise and from the image processing techniques used to extract and track features. When dealing with measurements of any sort, it is always the case that the measurements are accompanied by some error. Mistakes also occur, where mistakes are not large errors but failures of a system component or more. A description of errors, mistakes and modeling them can be found in [4,5].

The observer robot uses a camera to grab and register images of the manipulation system, thus, need to know the errors in mapping from the 3-D world features to the 2-D domain which is used in forming 3-D hypothesis about the task under supervision. The accuracy, precision and modeling uncertainty of the camera as our sensor is an important issue and the first step towards forming a full uncertainty model for recovering the 3-D events in the observer automaton.

As a lot of the image processing algorithms compute derivatives of the intensity function, noise in the image will be amplified and propagated throughout the observation process. The goal of this treatment is to find a distribution for the uncertainty of mapping a specific 3-D feature into a specific pixel value. In other words, if the feature 2-D position was discovered to be (i, j), then the goal is to find a 2-D distribution for i and j, assuming that there is no uncertainty in the technique used to extract the 2-D feature, the technique's uncertainty will be discussed in the next section. The end product of modeling the sensor uncertainty is to be able to say a statement like : "The 3-D feature F is located in the 2-D pixel position (i, j) with probability p_1 or located in the 2-D pixel position $(i, j+1)$ with probability p_2 or .... *given* that the registered location is (l, m), such that $p_1 + p_2 + ... + p_n = 1$, and $\nexists$ error in the 2-D feature recovery mechanism."

6.1 Image Formation Errors

The errors in the image formation process are basically of two different kinds, as was discussed in [5]. The first type is a spatial error, the other type is a temporal error. The spatial error due to the noise characteristics of a CCD transducer can be due to many reasons, among which are dark signatures and illumination signatures. The technique to be used is to take a large number of images, we can denote the image intensity function as a 3-D function $I(u, v, t)$, with spatial arguments u and v and temporal argument t. The sample mean of the image intensities over N time samples can be denoted by $\overline{I}(u, v)$.

$$\overline{I}(u, v) = \frac{1}{N} \sum_{t=1}^{N} I(u, v, t)$$

The spatial variance in a 5×5 neighborhood of the means is computed by:

$$s^2(u, v) = \sum_{i=-2}^{2} \sum_{j=-2}^{2} (\overline{I}(u+i, v+j) - \overline{I}(u, v))^2$$

The dark signature of the camera can be determined by computing $\overline{I}(u, v)$ of each pixel with the lens cap on. It will be found that a small number of pixels will have non-zero mean and non-zero variance. The specific pixel locations are blemished and should be registered. The uniform illumination is computed by placing a nylon diffuser over the lens and computing the mean and variance. It will be noticed that due to digitizing the CCD array into a pixel array of different size, and the difference in sample rates between the digitizer and camera, the border of the image will have different mean and variance from the interior of the image. Some "stuck" pixels at the location of the blemished pixels will also be noted. The contrast transfer function will also be noted to vary at different distances from the center of the lens.

Temporal noise characteristics can also be identified by taking a number of experiments and notice the time dependency of the pixels intensity function. In our treatment and for our modeling purposes we concentrate on the spatial distribution of noise and its effect on finding the 2-D uncertainty in recovering a 3-D feature location in the pixel array.

6.2 Calibration and Modeling Uncertainties

Methods to compute the translation and rotation of the camera with respect to its coordinates, as well as the camera parameters, such as the focal length, radial distortion coefficients, scale factor and the image origin, have been developed and discussed in the literature [10,28,48]. We use a static camera calibration technique to model the uncertainty in 3-D to 2-D feature locations. In particular we use the sequence of steps used to transform from 3-D world coordinates to computer pixel coordinates in order to recover the pixel uncertainties, due to the sensor noise characteristics described previously.

The inputs to the system we utilize are two sets of coordinates, (X_f, Y_f), which are the computer 2-D pixel image coordinates in frame memory and (x_w, y_w, z_w), which are the 3-D world coordinates of a set of coplanar points impressed on a piece of paper with known inter-point distances. A discussion of the exact mathematical formulation of the inter-step computations to find all the parameters can be found in [10]. Our approach is to treat the whole camera system as a black box and make input/output measurements and develop a model of its parametric behaviour. The next step is to utilize the recovered camera parameters and the number of 3-D points which we created in order to formulate a distribution of the 2-D uncertainty.

The strategy used to find the 2-D uncertainty in the features 2-D representation is to utilize the recovered camera parameters and the 3-D world coordinates (x_w, y_w, z_w) of the known set of points and compute the corresponding pixel coordinates, for points distributed throughout the image plane a number of times, find the actual feature pixel coordinates and construct 2-D histograms for the displacements from the recovered coordinates for the experiments performed. The number of the experiments giving a certain displacement error would be the z axis of this histogram, while the x and y axis are the displacement error. Different histograms can be used for different 2-D pixel positions distributed throughout the image plane. The three dimensional histogram functions are then normalized such that the volume under the histogram is equal to 1 unit volume and the resulting normalized function is used as the distribution of pixel displacement error, thus modeling the sensor uncertainty. The black box approach is thus used to model errors in a statistical sense.

7 Image Processing Uncertainties

In this section we describe a technique by which developing uncertainties due to the image processing strategy can be modeled. In addition, we end the discussion by combining both the sensor uncertainties developed in the previous section and the models developed in this section to generate distribution models for the uncertainty in estimating the 2-D motion vectors. These models are to be used for determining the full uncertainty in recovering the 3-D events that causes state transitions between states of the observer automaton.

We start by identifying some basic measures and ideas that are used frequently to recognize the behaviour of basic image processing algorithms and then proceed to describe the technique we use in order to compute the error model in locating certain features from their 2-D representation in the pixel array. We concentrate on modeling the error incurred in extracting edges, as edge extraction is a very popular mechanism that is used for both identifying feature points on the manipulating hand and also for computing 2-D contours of the object under supervision. When we discussed flow recovery techniques before, it was discussed in details that the optic flow recovery algorithm using local matching works well for the intensity boundaries and not for the inside regions.

7.1 Edge Extraction Uncertainties

Edge extraction strategies and methods to evaluate their performance qualitatively and quantatively have been presented and discussed in the literature [16,18,31,37]. There are many types of edges, ideal, ramp and noisy edges are only three of them. Different curvatures in the edges also constitute another dimension to be taken into consideration when it comes to asserting the types of edges that exist in an image.

The goal of developing the error models for edge extraction to to be able to say a statement like : "Given that the 2-D feature recovered using the edge recovery S is in pixel position (x, y), then there is a probability that the feature was originally at pixel position $(x+1, y)$ with probability p_1 or .... etc. due to the noise in the pixel image, such that $p_1 + p_2 + ... + p_n = 1$." The problem is to find the probabilities.

It should be obvious that there may be different types of noises and also different levels of those types that might vary at different locations in the sensor image plane. This adds to the different models that we might have to construct. We use ideal, that is, synthesized edges of different types, locations and also orientations in image frames then corrupt them with different kinds and levels of noises. We know the ideal edge points from the ideal image, for which we shall use the edge detector that is to be used in the observer

Figure 18: Distribution of the x-coordinate displacement

experiment. The corrupted images will then be operated upon by the detector and the edge points located. The edge points will differ from the ideal image edge points. The problem reduces to finding corresponding edge points in corrupted and ideal images then finding the error along a large number of edge points. A 2-D histogram is then constructed for the number of points with specific displacement errors from the ideal point. The volume of the histogram is then normalized to be equal to 1, the resulting 3-D function is the 2-D probability density function of the error of displacements. For practicality measures, the process can be repeated for orientations differing by 15^o and the set of distributions preserved. Whenever the observer automaton deals with a specific edge while extracting features, the corresponding distribution is referenced.

7.2 Computing 2-D Motion Uncertainty

In this section we describe how to combine sensor and image processing strategy error models to compute models for the recovered image flow values. To simplify the idea, let's assume that we have recovered a specific feature point (x_1, y_1) in an image grabbed at time instant t and the corresponding point (x_2, y_2) at time $t+1$. The problem is to figure out the distribution of v_x. As an example, to explain the procedure, let's assume that from the 3-D sensor distribution we have have computed the projection of the x coordinate of x_1 in the point:

$$f_X(x) = \int_R f_{X,Y}(x, y)dy$$

where R is all the possible y values within the sensor uncertainty model. The same process is applied for the strategy distribution and another function is recovered. To simplify things, lets assume that both distributions are identical to the distribution in Figure 18, that is, there is an equal probability equal to $\frac{1}{3}$ that the x coordinate is the same, or shifted one position to the left or the right. Combining the spatial information of both distributions as a convolution process would produce the distribution shown in Figure 19, which is the error probability density function of having the 3-D feature x 2-D coordinate in the recovered image 2-D x position. Further more, assume that x_2 distribution is the same.

The problem reduces to finding the distribution of the optic flow x component, using these two combined distributions. As an example, if $x_1 = 10$ and $x_2 = 22$, then all probability statements can be easily computed, a set of some of these probability statement is shown :

Figure 19: Combined sensor and strategy distribution of displacement

$$P(v_x = 8) = P((x_1 = 12) \wedge (x_2 = 20)) = \frac{1}{9} \times \frac{1}{9} = \frac{1}{81}$$

$$P(v_x = 9) = P(((x_1 = 12) \wedge (x_2 = 21)) \vee ((x_1 = 11) \wedge (x_2 = 20))) = (\frac{1}{9} \times \frac{2}{9}) + (\frac{2}{9} \times \frac{1}{9}) = \frac{4}{81}$$

$$P(v_x = 10 | x_1 = 10) = \frac{P(x_1 = 10 \wedge x_2 = 20)}{P(x_1 = 10)} = \frac{\frac{3}{9} \times \frac{1}{9}}{\frac{3}{9}} = \frac{1}{9}$$

Consequently, all distributions and expected values can be computed from the combination of the sensor level and strategy level uncertainty formulation. Those flow models are then passed to the higher levels for 3-D recovery. In the next section we discuss a method for refining the measured 2-D motion vectors and we then proceed to formulate the 3-D modeling of events as defined by the observer automaton.

8 Refining Image Motion

In this section we describe a method to refine the recovered 2-D motion vectors on the image plane. Having obtained from the sensor and extraction strategy uncertainty levels distribution estimates for the image flow of the different features, we now eliminate the unrealistic ones. We concentrate on the flow estimates for the motion of the manipulating hand and develop a technique that is to be used during the observation process as a means to reject faulty estimates. Faulty estimates can result from noise, errors or mistakes in the sensor acquisition process, manipulation or visual problems like occlusion, modeling the uncertainties in the previous two levels may still leave room for such anomalies.

We assume that the features to be tracked on the hand lie on a planar surface or that segmenting the hand as a polyhedra object into planar surfaces is simple, although the modification would be very simple to allow for arbitrary 3-D positions of the feature distribution. Since we know a-priori some information about the mechanical capabilities and limitations and geometric properties of the hand, also about the rate of visual sampling for the observer, we might be able to assert some limits on some of the visual parameters in our system.

Figure 20: Fitting Parabolic Curves

To illustrate the idea behind the approach, consider Figure 20, assume all the curves are 2-D parabolic functions $y = ax^2 + bx + c$, if the set of data points are as shown in the figure, then a least square error fit will produce the function D. However, if we know some upper and lower limits on the values of the coefficients a, b and c then we might be able to construct an upper and lower function parabolas A and C as an enclosing envelope, outside which we can reject all the data points. In that case, we can do a fit for the points that lie inside the envelope and obtain a more realistic function as shown by the curve B.

The situation for rejecting estimates of the image flow is not much different. We know equations that govern the behaviour of the image flow as a function of the structure and 3-D motion parameters, as follows :

$$v_x = (1 - px - qy)\left(x\frac{V_Z}{Z_o} - \frac{V_X}{Z_o}\right) + \left[xy\Omega_X - \left(1 + x^2\right)\Omega_Y + y\Omega_Z\right]$$

$$v_y = (1 - px - qy)\left(y\frac{V_Z}{Z_o} - \frac{V_Y}{Z_o}\right) + \left[\left(1 + y^2\right)\Omega_X - xy\Omega_Y - x\Omega_Z\right]$$

Which are second degree functions in x and y in three dimensions, $v_x = f_1(x, y)$ and $v_y = f_2(x, y)$.

In addition, we know upper and lower limits on the coefficients p, q, V_X, V_Y, V_Z, Ω_X, Ω_Y, Ω_Z and Z_o, as the mechanical abilities of the robot arm holding the hand will make the relative velocity and distance between the camera and hand impossible to exceed specific values within visual sampling timing period. So the problem reduces to constructing the three dimensional envelopes for v_x and v_y as the worst case estimates for the flow velocity and rejecting any measured values that lie outside that envelope. Figure 21 indicates the maximal and minimal v_x that can ever be registered on the CCD array of the camera, the x and y are in millimeters and the $x - y$ plane represents the CCD image plane, the depth Z is the maximal or minimal v_x in millimeters on the CCD array that can ever be registered. It can be noticed that they are symmetric due to the symmetry in the limits of the coefficients.

As an example, we write the equation governing the maximum v_x value in the first quadrant of the $x - y$ plane (x^+, y^+).

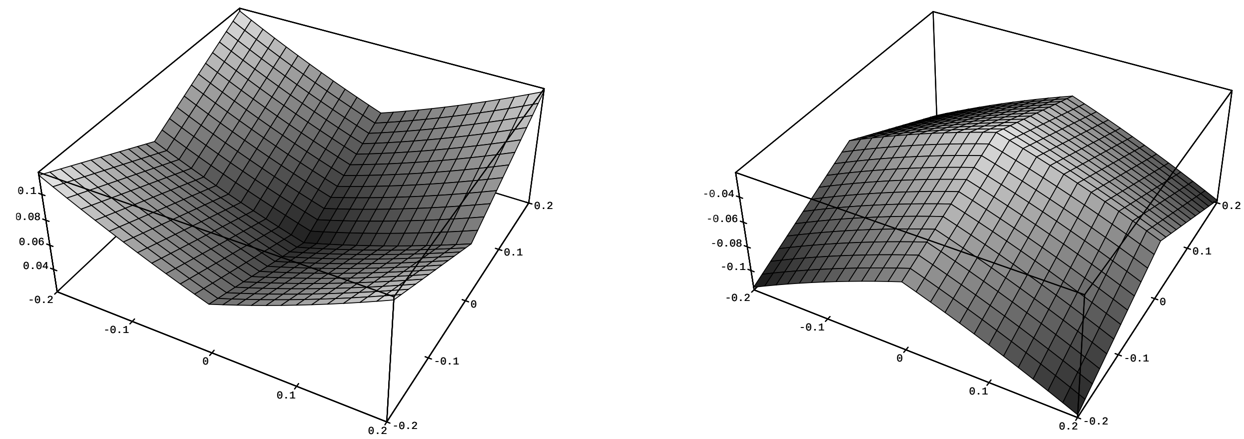


Figure 21: Maximal and Minimal v_x

$$v_{x_{max}} = \left(-\frac{fV_{X_s}}{Z_{o_s}} - f\Omega_{Y_s}\right) + \left(\frac{V_{Z_l}}{Z_{o_s}} + \frac{max(p_l V_{X_l}, p_s V_{X_s})}{Z_{o_s}}\right) x + \left(\frac{max(q_l V_{X_l}, q_s V_{X_s})}{Z_{o_s}} + \Omega_{Z_l}\right) y$$

$$+ \left(\frac{\Omega_{X_l}}{f} - \frac{min(q_l V_{Z_s}, q_s V_{Z_l})}{f Z_{o_s}}\right) xy - \left(\frac{min(p_l V_{Z_s}, p_s V_{Z_l})}{f Z_{o_s}} + \frac{\Omega_{Y_s}}{f}\right) x^2$$

where the subscripts s and l denote lower and upper limits, respectively. The problem of determining the maximum value of v_x seems to be a constrained non linear optimization problem, which is true, however, assuming that the upper and lower limits of the coefficients are equal in magnitude and opposite in directions (except for Z_o, which is used only as $Z_{o_s}^{+}$) makes the input to the max and min functions in the above equations always equal and thus providing one more degree of freedom in choosing the parameters and making the choice consistent throughout the equation. Thus the problem becomes simply to write eight equations as the above one for each of v_x and v_y and draw the function in each of the four quadrants for maximum and minimum envelopes. We shall not rewrite the sixteen equations here, but we show the results for v_x in Figure 21. It should be noted that the maximum absolute possible value of the image flow is minimal at the origin of the camera image plane and increases quadratically as the distance increases from the center.

The above envelopes are then used to reject unrealistic 2-D velocity estimates at different pixel coordinates in the image. As a further note, it should be mentioned that some on-line elimination procedures can be implemented depending on the current positions in the observer automaton, for example, the image flow field tends to assume certain configurations in the image plane depending on the 3-D motion, independent of the object's or the hand's structure, if the motion is only relative rotational velocities, the flow vectors all tend through pass from the same point. In other words, in addition to off-line a-priori estimation of the envelopes and on-line testing of measurements, we can also develop custom rejection techniques for certain observer automata states.

9 Recovering World Events

In this section we describe different techniques for recovering the 3-D events. In particular, we utilize the refined 2-D motion distributions that were computed in the previous levels in order to achieve a robust estimation of the three dimensional motion and structure vectors of the scene under observation. We develop some techniques for finding estimates of the required parameters and discuss mathematical formulations that will enable us to determine the 3-D event distributions. We concentrate in our treatment of the subject on determining the manipulating hand parameters, as the hand configuration is well defined, we also continue using the assumption that the feature points lie on a planar surface. As argued before, the extension to arbitrary configurations is straight forward. The object behaviour can be asserted using similar techniques and/or by observing conveniently located surface patches under similar assumptions.

The problem of recovering scene structure and the camera motion relative to the scene has been one of the key problems in computer vision. Many techniques have been developed for the estimation of structure and motion parameters (Tsai and Huang [47], Weng et al. [52] etc.). A lot of existing algorithms depend on evaluating the motion parameters between two successive frames in a sequence. However, recent research on structure and motion has been directed towards using a large number of frames to exploit the history of parametric evolution for a more accurate estimation and noise reduction (Ullman [50], Grzywacz and Hildreth[19] etc.)

We describe a method for recovering the 3-D motion and orientation of the planar surface (on which lies the hand features) from an evolving image sequence. The algorithm utilizes the image flow velocities in order to recover the 3-D parameters. First, we develop an algorithm which iteratively improves the solution given two successive image frames. The solution space is divided into three subspaces - the translational motion, the rotational motion and the surface slope. The solution of each subspace is updated by using the current solution of the other two subspaces. The updating process continues until the motion parameters converge, or until no significant improvement is achieved.

Second, we further improve the solution progressively by using a large number of image frames and the ordinary differential equations which describe the evolution of motion and structure over time. Our algorithm uses a weighted average of the expected parameters and the calculated parameters using the 2-frame iterative algorithm as current solution and continues in the same way till the end of the frame sequence. Thus it keeps track of the past history of parametric evolution.

The solution is further improved by exploiting the temporal coherence of 3-D motion. We develop the ordinary differential equations which describe the evolution of motion and structure in terms of the current motion/structure and the measurements (the 2-D motion vectors) in the image plane. As an initial step we assume that the 3-D motion is piecewise uniform in time. The extended Kalman filter can then be used to update the solution of the differential equations.

9.1 A 3-D Recovery Algorithm

One can model an arbitrary 3-D motion in terms of stationary-scene/moving-viewer as shown previously in Figure 17. The optical flow at the image plane can be related to the 3-D world as indicated by the following pair of equations (In case of a planar surface), for each point (x, y) in the image plane :

$$v_x = (1 - px - qy)\left(x\frac{V_Z}{Z_o} - \frac{V_X}{Z_o}\right) + \left[xy\Omega_X - \left(1 + x^2\right)\Omega_Y + y\Omega_Z\right]$$

$$v_y = (1 - px - qy)\left(y\frac{V_Z}{Z_o} - \frac{V_Y}{Z_o}\right) + \left[\left(1 + y^2\right)\Omega_X - xy\Omega_Y - x\Omega_Z\right]$$

where v_x and v_y are the image velocity at image location (x, y), (V_X, V_Y, V_Z) and $(\Omega_X, \Omega_Y, \Omega_Z)$ are the translational and rotational velocity vectors of the observer, p and q are the planar surface orientations. The situation becomes, for each point, two equations in eight unknowns, namely, the scaled translational velocities V_X/Z_o, V_Y/Z_o and V_Z/Z_o, the rotational velocities Ω_X, Ω_Y and Ω_Z and the orientations p and q. Differential methods could be used to solve those equations by differentiating the flow field and by using approximate methods to find the flow field derivatives. The existing methods for computing the derivatives of the flow field usually do not produce accurate results. Our algorithm uses a discrete method instead, i.e, the vectors at a number of points in the plane is determined and the problem reduces to solving a system of nonlinear equations.

It should be noticed that the resulting system of equations is nonlinear, however, it has some linear properties. The rotational part, for example, is totally linear, also, for any combination of two spaces among the rotational, translational and slope spaces, the system becomes linear. For the system of equations to be consistent, we need the flow estimates for at least four points, in which case there will be eight equations in eight unknowns.

9.1.1 Two-Frame Algorithm

The algorithm takes as input the estimate of the flow vectors at a number of points ≥ 4 obtained from motion between two images. It iterates updating the solution of each subspace by using the solution of the other two subspaces. Each update involves solving a linear system, thereby it requires to solve three linear systems to complete a single iteration. This process continues until the solution converges, or until no significant improvement is made. The algorithm proceeds as follows :

1. Set p, $q = 0$;
   input the initial estimate for rotation ;
   Solve the linear system for translation;
2. Use the translation and rotation from step 1 ;
   Solve the linear system for the slope ;
3. Set i=1;
   While (i $\leq$ Max. Iterations) and (no convergence) Do
      Solve for the rotations using latest estimates of translations, p and q;
      Solve for the translations using latest estimates of rotations, p and q;
      Solve for p, q using latest estimates of translations and rotations;
   end While ;

9.1.2 Complexity and Observations

As we mentioned earlier, one should notice in the equations relating the flow velocities with the slope, rotational and translational velocities that they are "quasi-linear" , if one can say so. The equations exhibit some linear properties. This suggests that a purely iterative technique for solving non-linear equations might not be an excellent choice, since, the variables are linearly related in some way. To think of a way of "inverting" the relations might be a good start, although to do that without a framework based on iterating and gravitating towards a solution is not a good idea.

Thus, we apply a method which converges faster than a purely iterative scheme like Newton's method. The complexity of Newton's method is determined by the complexity of computing the inverse Jacobian, which is of an order of N^3, or $N^{2.81}$ multiplications as the lower bound using Strassen's technique. In our case, since there is at least 8 equations in 8 unknowns, the complexity is of order $8^3 = 512$ multiplications at every iteration, and the method does not make any use of the fact that the set of equations at hand exhibits some linear properties.

The algorithm proposed, on the other hand, exploits the linearity in the equations, by inverting the set of relations for each subspace at every iteration. The complexity at every iteration is of the order of the complexity of computing the pseudo-inverse which is of the order of ($3^3 + 3^3 + 2^3$) multiplications at each iteration. This is equal to 62 multiplications at every iteration, which is significantly less than the 512 multiplications in a method like Newton's for example. It was noticed that the algorithm converged to solution in a very small number of iterations for most experiments we have conducted so far. The maximum number of iterations was 6.

Using the latest solution obtained from the two-frame analysis as the initial condition for the next two-frame problem in the image sequence would further decrease the complexity, as the next set of parameters would, most probably, be close in values to the current parameters, thus the number of iterations needed to converge to the new solution would decrease significantly.

The algorithm is not sensitive to the initial condition of the orientation parameters. The plane is simply assumed to be a frontal one at the beginning. The slope parameters evolves with iterations. It was noticed that the algorithm performs better for a large number of points that are evenly distributed throughout the planar surface, than it does for clustered, smaller number of image points. It is proven that there exists dual solutions for such systems. However, if our method gravitates towards a "fixed point" in the solution space we can find the other explicitly in terms of the first one from the relations given by Waxman and Ullman [51].

9.1.3 Multi-Frame Algorithm

The ordinary differential equations that describe the evolution of motion and structure parameters are used to find the expression for the expected parameter change in terms of the previous parameter estimates. The expected change and the old estimates are then used to predict the current motion and structure parameters.

At time instant t, the planar surface equation is described by

$$Z = pX + qY + Z_o$$

To compute the change in the structure parameters during the time interval dt, we differentiate the above equation to get

$$\frac{dZ}{dt} = p\frac{dX}{dt} + X\frac{dp}{dt} + q\frac{dY}{dt} + Y\frac{dq}{dt} + \frac{dZ_o}{dt}$$

The time derivatives of (X, Y, Z) in the above expression are given by the three components of the vector $-(\mathbf{V} + \mathbf{\Omega} \times \mathbf{R})$ that represent the relative motion of the object with respect to the camera. Substituting these components for the derivatives and the expression $pX+qY+Z_o$ for Z we can get the exact differentials for the slopes and Z_o as

$$\begin{aligned} dZ_o &= Z_o \left[(\Omega_Y + V_X)p - (\Omega_X - V_Y)q - V_Z\right] dt \\ dp &= \left[p(\Omega_Y p - \Omega_X q) + (\Omega_Y + \Omega_Z q)\right] dt \\ dq &= \left[q(\Omega_Y p - \Omega_X q) - (\Omega_X + \Omega_Z p)\right] dt \end{aligned}$$

Using the above relations, we can compute the new structure parameters at time $t + dt$ as

$$\acute{p} = p + dp \ , \ \acute{q} = q + dq \ \text{ and } \ \acute{Z_o} = Z_o + dZ_o$$

Thus the slope parameters evolve at time $t + dt$ as follows :

$$\begin{bmatrix} \acute{p} \\ \acute{q} \end{bmatrix} = \begin{bmatrix} p \\ q \end{bmatrix} + \begin{bmatrix} \Omega_Y p - \Omega_X q & \Omega_Z & \Omega_Y \\ -\Omega_Z & \Omega_Y p - \Omega_X q & -\Omega_X \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix} dt$$

The new translational velocity $\acute{V}$ at time $t + dt$ can be found in the absence of accelerations from

$$\acute{\mathbf{V}} = \mathbf{V} + \mathbf{V} \times \mathbf{\Omega} dt$$

Dividing $\acute{V}$ by $\acute{Z_o}$ we get the new expected scaled translational velocity components at time $t + dt$ as follows :

$$\begin{bmatrix} \acute{V_X} \\ \acute{V_Y} \\ \acute{V_Z} \end{bmatrix} = \begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} + \begin{bmatrix} -s & \Omega_Z & \Omega_Y \\ -\Omega_Z & -s & \Omega_X \\ \Omega_Y & -\Omega_X & -s \end{bmatrix} \begin{bmatrix} V_X \\ V_Y \\ V_Z \end{bmatrix} dt,$$

where s is expressed as follows :

$$s = (\Omega_Y + V_X)\,p - (\Omega_X - V_Y)\,q - V_Z$$

The expected rotational parameters at time $t + dt$ remain equal to their values at time t since

$$\acute{\mathbf{\Omega}} = \mathbf{\Omega} + \mathbf{\Omega} \times \mathbf{\Omega} dt = \mathbf{\Omega}$$

and thus

$$\left(\acute{\Omega_X}, \acute{\Omega_Y}, \acute{\Omega_Z}\right) = (\Omega_X, \Omega_Y, \Omega_Z)$$

Figure 22: Two-Frame Algorithm

Our first multi-frame algorithm uses a weighted average of the expected parameters at time $t + dt$ from the above equations and the calculated parameters using the two-frame iterative algorithm as the solution at time $t + dt$, and continues in the same way until the end of the frame sequence. Thus it keeps track of the past history of parametric evolution. We further develop the first multi-frame algorithm to exploit the temporal coherence of 3-D motion. We develop the ordinary differential equations which describe the evolution of motion and structure in terms of the current motion/structure and the two-dimensional flow vectors in the image plane. We assume that the 3-D motion is piecewise uniform in time, i.e, $\dot{\vec{\mathbf{\Omega}}} = \dot{\vec{\mathbf{V}}} = \mathbf{0}$. We then use the equations expressing the time derivative of the slope derived above and the fact that the derivative of the rotational velocities is zero and develop the following expressions for the scaled translational velocities and the depth Z_o :

$$\frac{d\acute{V}_X}{dt} = -\acute{V}_X \frac{1}{Z_o}\frac{dZ_o}{dt}, \; \frac{d\acute{V}_Y}{dt} = -\acute{V}_Y \frac{1}{Z_o}\frac{dZ_o}{dt} \quad \text{and} \quad \frac{d\acute{V}_Z}{dt} = -\acute{V}_Z \frac{1}{Z_o}\frac{dZ_o}{dt}$$

$$\frac{1}{Z_o}\frac{dZ_o}{dt} = -\acute{V}_Z - p v_x - q v_y$$

The extended Kalman filter is then used to update the solution of the differential equations. Where the state vector can be written as :

$$X = [\; \acute{V}_X \;\; \acute{V}_Y \;\; \acute{V}_Z \;\; \Omega_X \;\; \Omega_Y \;\; \Omega_Z \;\; p \;\; q \;]$$

and the measurement vector is expressed as :

$$Z = [\; v_x \;\; v_y \;\; \frac{\delta v_x}{\delta x} \;\; \frac{\delta v_y}{\delta x} \;\; \frac{\delta v_x}{\delta y} \;\; \frac{\delta v_y}{\delta y} \;\; \frac{\delta v_x}{\delta t} \;\; \frac{\delta v_y}{\delta t} \;]$$

The behaviour of the two-frame algorithm and the multi-frame algorithm can be conceptualized as a control system as shown in Figures 22 and 23. Parallel implementations could be designed for the system, thus solving for the structure - motion parameters for each surface separately. In fact, solving the linear system at each iteration could also be parallelized. Extra processing is needed to segment the polyhedra-like hand into separate planar surfaces.

Figure 23: Multi-Frame Algorithm

9.2 Other Algorithms

There are other non-iterative techniques for recovering the 3-D parameters resulting from 2-D motion between two frames. The methods that will be mentioned here rely on specific assumption regarding the hand's geometry and/or world manipulating actions. Assuming that the actual relations between feature points that lie on the hand plane is well defined than a closed form solution for the structure parameters and depth can be estimated by using a method like the one described by Fischler and Bolles [17]. The motion parameters can then be easily recovered by solving a linear system in six parameters.

It should be noticed that we use alternative methods in order to solve linear equations at different automaton states, the motive behind that is the fact that linear systems can be solved in a pseudo-real time framework for a relatively small number of feature points and in addition a closed form solution always results. Another idea is to assume that the surface of the manipulating hand is frontal at the time of capturing the frame to be processed with the previous one, thus p and q are equal to zero, and the problem reduces to solving a linear system in six parameters for the motion parameters, while the depth is easily computed by knowing the 3-D distance between any two feature points.

The assumption here being that the observer always locates itself to a position in which the hand is frontal with respect to the camera image plane, and that manipulating movements while the camera is moving and during computations is negligible. Other formulations may attempt to find pseudo-close form solution of the non-linear second order system and other assumptions, like the absence of rotational and/or translational motion reduces the complexity significantly.

9.3 Recovering 3-D Uncertainties

Having discussed methods for computing the three dimensional motion vectors and structure parameters between two image frames, we now use the same formulations described earlier for 3-D recovery, but using 2-D error distributions as estimates for motion and/or feature coordinates in order to compute 3-D uncertainty distributions for the real world motion vectors and structure instead of singular values for the world events.

As an example to illustrate the idea, let's assume that we have a linear system of equations as follows :

$$x + 3y = z_1$$

$$2x + y = z_2$$

The solution of this system is very easily obtained as :

$$x = \frac{3}{5}z_2 - \frac{1}{5}z_1$$

$$y = \frac{2}{5}z_1 - \frac{1}{5}z_2$$

That is, a linear combination of the right hand side parameters. If the parameters z_1 and z_2 were random variables of known probability distributions instead of constants, then the problem becomes slightly harder, which is, to find the linear combination of those random variables as another random variable. The obvious way of doing this would be to use convolutions and the formula :

$$P_{X_1+X_2}(y) = \sum_R P_{X_1,X_2}(x, y - x)$$

for the sum of two random variables X_1, X_2 for any real number y and/or the formula for linear combinations over the region R, which is for all x such that $P_{X_1,X_2}(x, y - x) > 0$. Using the moment generating function or the characteristic function seems also to be a very attractive alternative. The moment generating function M of a linear combination of random variables, for example X_1, X_2 can be written as :

$$M_{aX_1+bX_2+c}(t) = e^{ct}\left(M_{X_1}(at)M_{X_2}(bt)\right)$$

for independent random variables X_1, X_2. That is, the problem of solving linear systems on the form $Ax = b$, where b is a vector of random variables, may be reduced to finding closed form solutions for x in terms of the random parameters (using any elimination technique) and then manipulating the results and finding different expectations using moment generating or characteristic functions.

The solutions we suggest to this problem of finding the random variable distributions of the 3-D parameters utilize the techniques we described in the previous two subsections. Using either the two-frame iterative technique or the closed form algorithms, it should be noticed that the problem reduces to either solving multi-linear systems or a single one. In that case, using elimination and characteristic functions for computing the required expectations and distributions is straight forward. As an example, the recovered 3-D translational velocity cumulative density functions for an actual world motion equal to :

$$V_X = 0\ cm,\ V_Y = 0\ cm \text{ and } V_Z = 13\ cm$$

is shown in Figure 24. It should be noted that the recovered distributions represents a fairly accurate estimation of the actual 3-D motion.

Thus, we have suggested algorithms for the quick estimation of the 3-D uncertainties in the structure and motion of the manipulation system. The next step would be to refine these estimates and use them for asserting the world events. This will be described in the following two sections.

Figure 24: CDF of V_X, V_Y, and V_Z

10 Refining World Events

In this section we describe techniques for eliminating and refining the 3-D models of manipulation under observation, whose recovery was discussed in the previous sections. In particular, we discuss a strategy to reject improbable events that might have been computed due to noise and uncertainties that were not compensated for in the distribution formulation, also because of unsmooth visual artifacts. We employ both existing knowledge about the mechanical properties of the manipulation and also knowledge from the current state of the observer automaton.

The hand is assumed to be a well defined entity, changing the hand and/or its characteristics can be modeled by simply plugging in a module that describes the new characteristics, the *same* hand is used through out the entire manipulation activities. Knowing the joint limits of the manipulating robot will enable us to reject improbable recovered 3-D motion vectors, that could not have occurred in the real 3-D world. As an example, assuming that we use a gripper with two "claws" having only one degree of freedom, then, obviously, any recovered 3-D rotational velocities for the claws should be rejected. Unrealistic slope estimations should also be rejected.

The current position in the observer automata will allow refining the recovered 3-D event distributions, as it might well be the case that impossible manipulation actions at a specific manipulation stage are recovered. It is impossible, for example, due to the visual sampling rate, that the hand is in and upright position holding a nail in the center of the image plane at a time step, then having it disappear or hold another object at a distant 3-D position in the next time step, unless, of course a manipulation or viewer system failure has happened. In that case, some designated fail state should be accessed, discarding the recovered parameters. Limits on V_X, V_Y, V_Z, Ω_X, Ω_Y, Ω_Z and Z are asserted for every observer subset of states, and used for refining the recovered 3-D world events.

11 Navigating the Observer Automaton

At this point in the hierarchy of recovery and uncertainty levels, we have established methods and algorithms for recovering the refined three dimensional velocity and structure of the scene under observation. In addition, we computed the distribution of the uncertainty in the numerical values of the parameters in real-time. For example, the computed value for the translational velocity V_X might be a random variable lying between two values V_1 and V_2 with a known probability distribution $\mathcal{F}$. The same applies for all the other parameters for the different components in the scene.

The problem now is how to make use of these distributions in order to navigate the observer automaton as defined in section 2 and demonstrated by examples in section 3. In other words, having built the DEDS automaton model of the visual system and its observer, we have a set of events that are defined as ranges on the visual scene parameters that causes state transitions between the automaton states. As a simple example, there might be two different events branching from a state in some task observer automaton and causing state transitions to two other states, and a self loop caused by the continuous dynamics within a coarse quantization of a DEDS state, as follows :

$$S_1 \xrightarrow{e_1 : \Omega_1 < \Omega_Y < \Omega 2} S_2$$

$$S_1 \xrightarrow{e_2 : -\Omega_1 < \Omega_Y < \Omega 1} S_1$$

$$S_1 \xrightarrow{e_3 : -\Omega_2 < \Omega_Y < -\Omega 1} S_3$$

In addition to other limits on the other scene parameters. That is, if Ω_Y occurs within a specific range, then the corresponding state transition should be asserted according to the above set of event description.

The problem then reduces to computing the corresponding areas under the refined distribution curves obtained from the hierarchy levels. In the case of the presence of more than a single parameter in the transition event description, then the corresponding area under *each* parameter curve should be computed and multiplied for each parameter in the event definition. The goal is to find the probability of the occurrence of each event. In the above example, the goal would be to find the probability of e_1, e_2 and e_3.

An obvious way of using those probability values is to establish some threshold values and assert transitions according to those thresholds. For example, if for any event in the set (e_1, e_2 and e_3), the computed probability of the range is > 0.7, then the corresponding state transition should be asserted. It should be noted that those threshold values are highly task and state-dependent, appropriate values for the thresholds can be determined by performing many experiments for different task descriptions. The thresholds can also be updated adaptively according to the current manipulation patterns under observation. Many problems may arise after having obtained the above probabilities at the current automaton state. It might be the case that none of the obtained probability values exceeds the set threshold value and/or all values are very low. In that case, there is a good chance that we are at either the wrong automata state, or that a gross error has occurred in manipulation or some system failure.

The remedy to such problems can be implemented through time proximity, that is, wait for a while (which is to be preset) till a strong probability value is registered and/or *backtrack* in the automaton model for the observer till a high enough probability value is asserted, a fail state is reached or the initial ambiguity is asserted. The backtracking strategy can be implemented using a stack-like structure associated with each state that has already been traversed. A stack of the latest computed probability values sorted in descending order as an index to the corresponding event. As soon as a forward traversal is performed, the top value should be popped. Backtracking can be done by using the top of the stack value and do the corresponding transition and compute the new probabilities for the events. A father state parameter should also accompany each state that has been already been traversed. In case all the stack has been exhausted for a specific state, the father state should be accessed and a new route be accessed. Exhausted states are labeled and never revisited while backtracking. For states that have not been visited at all, new stacks and computations should be be performed.

Having established techniques for navigating the observer, the model description is now completed. The formulation uses uncertainties to assert current states of the manipulation system and attempts to recover from mistakes and errors. The model uses different intermediate levels for computing uncertainties, from the sensor level to the observer automaton level.

12 Conclusions

We have proposed a new approach to solving the problem of observing an agent. In particular, we described a system for observing a manipulation process. Our approach uses the formulation of discrete event dynamic systems (DEDS) as a high-level model for the framework of evolution of the visual relationship over time. The proposed system utilizes the a-priori knowledge about the domain of the manipulation actions in order to achieve efficiency and practicality. The dynamic recursive context for finite state machines (DRFSM) was also introduced with some applications from the inspection and reverse engineering domains. The high level formulation allows for *recognizing* and *reporting* the visual system state as a symbolic description of the observed tasks.

The proposed formulation takes into consideration the presence of uncertainties in the observed behaviour of the system. The uncertainties are utilized in order to achieve robustness and to allow for correcting the observer's actions. Asserting transitions within the state description of the visual tasks is based on the recovered values of the observed parameters and the associated world uncertainties. The process develops coarse quantization of the visual actions in order to attain an active, adaptive and goal-directed sensing mechanism. Discrete aspects of the observation process are exploited in order to attach a meaningful symbolic interpretation of the observed task at different instances of the visual process. The formulation is flexible, since the quantization thresholds between different states can be tuned as the observed task requires. Continuous aspects of the process are also preserved as the relevant parameters are observed as the agent moves.

We started by describing the automaton model of a discrete event dynamic system then proceeded to formulate the frameworks, and the observer construction mechanisms. We develop efficient low-level event-identification mechanisms for determining different manipulation movements in the system and for moving the observer. Next, we define and construct six different levels for converting the raw 2-D image data into meaningful 3-D descriptions of the world events. The formulation includes computing uncertainty models resulting from errors in the 2-D and 3-D recovery mechanisms. The formulation allows the observer to navigate in real time with a stable behaviour through the automaton state space and thus assert world events and transitions.

The approach used can be considered as a framework for a variety of visual tasks, as it lends itself to be a practical and feasible solution that uses existing information in a robust and modular fashion. The work examines closely the possibilities for errors, mistakes and uncertainties in the manipulation system, observer construction process and event identification mechanisms. Ambiguities are allowed to develop and are resolved after finite time, recovery mechanisms are devised too. Theoretical and experimental aspects of the work supports adopting the framework as a new kind of basis for performing many task-oriented recognition, inspection and observation of visual phenomena. In the next section we examine extension ideas and future research opportunities for which the formulation can be considered as the backbone.

13 Extensions and Future Research

The proposed formulation can be extended to accommodate for more manipulation processes. Increasing the number of states and expanding the events set would allow for a variety of manipulating actions. The system can be made more "modular" by constructing a general automaton model of a discrete event dynamic system and defining the states, events and the certainty thresholds for them in an *automatic* way through a *learning* stage [32]. In other words, different manipulation actions can be performed and "shown" to the observer and then the possible states, events and sequences of operations are automatically embedded in the general dynamic model. Thus, the manual formulation of the DEDS model for the task would not be needed anymore.

More powerful models for the DEDS could be sought, for example, context sensitive grammars, push-down automata, Turing machines and/or μ-recursive functions. The model building process can be thought of as forming a compiler with the object, sensor, task description and learning model as inputs, and the algorithm to follow the observer automaton with uncertainty as the output. Feedback can be supplied to the manipulating system in order to correct its actions, thus closing the vision-manipulation loop. The system could be generalized to an arbitrary number of mobile manipulating robots and mobile observing ones, a scheme would have to be devised to allow for distributed and parallel control of the observation and feedback process in an efficient way and to prevent deadlock and/or starvation problems.

The characteristics of the workspaces of both the manipulating robot and the observer can be utilized in order to avoid problems like collision and occlusion. This might be necessary to explore if both workspaces intersect in a 3-D volume. This can occur in a simple laboratory setup with two fixed manipulators, visualizing the volume of intersection and the holes and voids [1] within each robot reachable workspace will be necessary for planning and constructing the model and its observer.

Foveal and peripheral vision strategies can be applied to "focus" on a specific aspect of the scene under considerations, according to the present observer state. Pyramid approaches for locating actions can be used. Logarithmic sensors, like cameras whose CCD array resembles the human eye can be utilized as the observer's visual sensor for shifting attention to the interesting parts of the image.

Parallelizing the whole process by forming simultaneous observers can be explored. This will be necessary in case of multiple observing robots, manipulating robots and/or different kinds of sensors (tactile, range, vision ..etc) so as to allow for modular and efficient planning, "seeing" and recovery mechanisms.

Inter-parallelization of different algorithms should be explored too. Overcoming delays in communication links between different observers and between the vision, control and parallelization modules within the same observer module should be addressed, specially if the modules are *physically* distant within the laboratory setup. Overcoming delays when feedback is supplied to the manipulating hand would be necessary.

The idea of DRFSM DEDS as skeletons for observation under uncertainty can be explored further to allow for various other visual tasks. We discussed observing manipulation as a subset of observing moving agents, however, similar formulation can be described for other tasks, like recognizing stationary objects with optimal observation costs, i.e, minimal motion events. Perturbation analysis [24,45] can be performed for the average task behaviour of frequent visual events within a specified manipulation domain. Disappearing objects and partially occluded objects can also be recognized optimally using the proposed scheme, using time proximity as another dimension for asserting the identity of different targets, that is, allow recognition and/or tracking to be completed within a pre-specified, task-dependent time frame.

References

[1] T. Alameldin, "Visualization of 3-D Workspaces", Ph.D. Thesis, Computer and Information Science Department, University of Pennsylvania, 1991.

[2] J. Aloimonos and A. Bandyopadhyay, "Active Vision". In *Proceedings of the* 1^{st} *International Conference on Computer Vision*, 1987.

[3] P. Anandan, "A Unified Perspective on Computational Techniques for the Measurement of Visual Motion". In *Proceedings of the* 1^{st} *International Conference on Computer Vision*, 1987.

[4] H. L. Anderson, *GRASP lab. Camera Systems and Their Effects on Algorithms*, Technical Report MS-CIS-88-85 and GRASP lab. TR 161, University of Pennsylvania, 1988.

[5] R. Bajcsy, E. Krotkov and M. Mintz, *Models of Errors and Mistakes in Machine Perception*, Technical Report MS-CIS-86-26 and GRASP lab. TR 64, University of Pennsylvania, 1986.

[6] R. Bajcsy, "Active Perception", *Proceedings of the IEEE*, Vol. 76, No. 8, August 1988.

[7] R. Bajcsy and T. M. Sobh, *A Framework for Observing a Manipulation Process.* Technical Report MS-CIS-90-34 and GRASP Lab. TR 216, University of Pennsylvania, June 1990.

[8] R. Bajcsy and T. M. Sobh, *Observing a Moving Agent.* Technical Report MS-CIS-91-01 and GRASP Lab. TR 247, Computer Science Dept., School of Engineering and Applied Science, University of Pennsylvania, January 1991.

[9] J. L. Barron, A. D. Jepson and J. K. Tsotsos, "The Feasibility of Motion and Structure from Noisy Time-Varying Image Velocity Information", *International Journal of Computer Vision*, December 1990.

[10] N. M. Benahmed, *Camera Calibration for Dynamic Environment.* M.S. Thesis, Department of Electrical Engineering, University of Pennsylvania, 1989.

[11] T. O. Binford, *Generic Surface Interpretation : Observability Model*, Technical Report, Robotics Laboratory, Stanford University, 1991.

[12] P. J. Burt, C. Yen, and X. Xu, "Multiresolution Flow-Through Motion Analysis". In *Proceedings of the 1983 IEEE Conference on Computer Vision and Pattern Recognition.*

[13] P. J. Burt, et al., "Object Tracking with a Moving Camera", *IEEE Workshop on Visual Motion*, March 1989.

[14] 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.

[15] M. S. Clark, "Robot-based Real-time Motion Tracker". In *Proceedings of the* 2^{nd} *SPIE Conference on Sensor Fusion*, November 1989.

[16] E. S. Deutsch and J. R. Fram, "A Quantitative Study of the Orientation Bias of some Edge Detector Schemes", *IEEE Trans. Comput.*, C-27, No. 3, March 1978.

[17] M. Fischler and R. C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, Readings in Computer Vision, Morgan Kaufmann Publishers, 1987.

[18] J. R. Fram and E. S. Deutsch, "On the Quantitative Evaluation of Edge Detection Schemes and Their Comparison with Human Performance", *IEEE Trans. Comput.*, C-24, No. 6, June 1975.

[19] N. M. Grzywacz and E. C. Hildreth, *The Incremental Rigidity Scheme for Recovering Structure from Motion: Position vs. Velocity Based Formulations*, MIT A.I. Memo No. 845, October 1985.

[20] D. J. Heeger, *Models for Motion Perception.* Ph.D. Thesis, Computer and Information Science Department, University of Pennsylvania, September 1987.

[21] J. Heel, "Dynamic Motion Vision", In *Proceedings of the SPIE Conference on Computer Vision*, November 1989.

[22] J. Hervé, P. Cucka and R. Sharma, "Qualitative Visual Control of a Robot Manipulator". In *Proceedings of the DARPA Image Understanding Workshop*, September 1990.

[23] J. Hervé, R. Sharma and P. Cucka, "Qualitative Coordination of a Robot Hand/Eye System". CAR–TR–516, Center for Automation Research, University of Maryland, 1990.

[24] Y. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems", *IEEE Transactions on Automatic Control*, July 1987.

[25] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[26] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow", *Artificial Intelligence*, vol. 17, 1981, pp. 185-203.

[27] B. K. P. Horn, *Robot Vision*, McGraw-Hill, 1987.

[28] A. Izaguirre, P. Pu and J. Summers, "A New Development in Camera Calibration: Calibrating a Pair of Mobile Cameras", In *Proceedings of the International Conference on Robotics and Automation*, pp. 74-79, 1985.

[29] D. Keren, S. Peleg and A. Shmuel, *Accurate Hierarchical Estimation of Optic Flow*, TR–89–9, Department of Computer Science, The Hebrew University of Jerusalem, June 1989.

[30] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1979.

[31] E. Krotkov, *Results in Finding Edges and Corners in Images Using the First Directional Derivative*, Technical Report MS-CIS-85-14 and GRASP lab. TR 37, University of Pennsylvania, 1985.

[32] Y. Kuniyoshi, M. Inaba and H. Inoue, *Teaching by Showing : Generating Robot Programs by Visual Observation of Human Performance*, Department of Mechanical Engineering, The University of Tokyo, Technical Report, 1990.

[33] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.

[34] Y. Li and W. M. Wonham, "Controllability and Observability in the State-Feedback Control of Discrete-Event Systems", *Proc.* 27^{th} *Conf. on Decision and Control*, 1988.

[35] H. C. Longuet-Higgins and K. Prazdny, *The interpretation of a moving Retinal Image*, Proc. Royal Society of London B, 208, 385-397.

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

[37] T. Peli and D. Malah, "A Study of Edge Detection Algorithms", *Computer Graphics and Image Processing*, vol. 20, 1982, pp. 1-21.

[38] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *SIAM Journal of Control and Optimization*, January 1987.

[39] P. J. Ramadge and W. M. Wonham, "Modular Feedback Logic for Discrete Event Systems", *SIAM Journal of Control and Optimization*, September 1987.

[40] G. E. Révész, *Introduction to Formal Languages*, McGraw-Hill, 1985.

[41] S. W. Lee and K. Wohn, *Tracking Moving Objects by a Robot-held Camera Using a Pyramid-Based Image Processor*, Technical Report MS-CIS-88-97 and GRASP Lab. TR 168, University of Pennsylvania, 1988.

[42] T. M. Sobh and R. Bajcsy, "A Model for Observing a Moving Agent". Will appear in *Proceedings of the Fourth International Workshop on Intelligent Robots and Systems (IROS '91)*, Osaka, Japan, November 1991.

[43] T. M. Sobh and K. Wohn, "Recovery of 3-D Motion and Structure by Temporal Fusion". In *Proceedings of the* 2^{nd} *SPIE Conference on Sensor Fusion*, November 1989.

[44] M. Subbarao and A. M. Waxman, *On The Uniqueness of Image Flow Solutions for Planar Surfaces in Motion*, CAR–TR–113, Center for Automation Research, University of Maryland, April 1985.

[45] Rajan Suri, "Perturbation Analysis : The State of the Art and Research Issues Explained via the GI/G/1 Queue", *Proc. of the IEEE*, January 1989.

[46] C. Tomasi and T. Kanade, "Shape and Motion without Depth", CMU-CS-90-128, School of Computer Science, Carnegie Mellon University, May 1990.

[47] R. Y. Tsai and T. S. Huang, "Estimating three-dimensional motion parameters of a rigid planar patch", *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-29(6), December 1981.

[48] R. Y. Tsai, "An Efficient and Accurate Camera Calibration Technique for 3-D Machine Vision", IBM Report.

[49] S. Ullman, "Analysis of Visual Motion by Biological and Computer Systems", *IEEE Computer*, August 1981.

[50] S. Ullman, *Maximizing Rigidity: The incremental recovery of 3-D structure from rigid and rubbery motion*, AI Memo 721, MIT AI lab. 1983.

[51] A. M. Waxman and S. Ullman, *Surface Structure and 3-D Motion From Image Flow: A Kinematic Analysis*, CAR–TR–24, Center for Automation Research, University of Maryland, October 1983.

[52] J. Weng, T. S. Huang and N. Ahuja, "3-D Motion Estimation, Understanding and Prediction from Noisy Image Sequences", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(3), May 1987.

[53] R. Wilson and G. H. Granlund, "The Uncertainty Principle in Image Processing", IEEE Trans. PAMI, Vol. 6, No. 6, November 1984.

[54] K. Wohn and S. R. Maeng, "Real-Time Estimation of 2-D Motion for Object Tracking", In *Proceeding of the SPIE Conference on Intelligent Robotics*, November 1989.