A Framework for Observing a Manipulation Process

Ruzena Bajcsy
University of Pennsylvania

Tarek Sobh
University of Pennsylvania

A Framework for Observing A Manipulation Process

MS-CIS-90-34
GRASP LAB 216

Ruzena Bajcsy
Tarek Sobh

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104

June 1990

A Framework for Observing a Manipulation Process

Ruzena Bajcsy and Tarek Sobh

GRASP Laboratory
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania, Philadelphia, PA 19104

Abstract

We propose a system for observing a robot hand manipulating an object. A discrete event dynamic system is used as a model for the manipulation process. A framework for the hand/object relationship is developed and a stabilizing observer is constructed for the system. We describe low-level modules for recognizing the "events" that causes state transitions within the dynamic manipulation system. Our system uses different tracking mechanisms in order to control the observation process in an efficient and stable manner.

Keywords : Computer Vision, Control Theory, Discrete Event Dynamic Systems, Finite State Automata, Image Processing, Manufacturing Systems, Robotics.

1 Introduction

The process of observing a robot hand manipulating an object is very crucial for many robotic and manufacturing tasks. It is important to know in an automated manufacturing environment whether the robot hand is doing the correct sequence of operations on an object (or more than one object). It might be a fact that the workspace of the robotic manipulator cannot be accessed by humans, as in the case of some space applications or some areas within a nuclear plant, for example. In that case, having another robot "look" at the process is a very good option. Thus, the observation process can be thought of as a stage in a closed-loop fully automated system where there are robots who perform the required manipulation task and some other robots who observe them and correct their actions when something goes wrong. Typical manipulation processes include grasping, pushing, pulling, lifting, screwing and unscrewing. Visual information from the observing robots can be the only kind of feedback, or it can be supplemented by other kinds, like tactile sensing. In this paper, we address the problem of observing a single hand manipulating a single object and "knowing" what is the hand doing, no feedback will be supplied to the manipulating robot to correct its actions.

To be able to observe how a hand manipulates an object, we must be able to identify how the hand moves and how the hand/object 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 [2,5,8,9,15], and to recover the three-dimensional world structure and motion [4,20,21,22,24,25,26]. Those techniques are not problem-oriented, they are not restricted to a particular problem domain, as is the case with our observer construction problem.

Trying to use the above techniques directly to solve our 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 manipulation process increases. Due to the fact that we probably know a-priori some information about the allowable (or useful) manipulation processes, 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 manipulation 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 [1,3], 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 a high-level structuring technique to model the manipulation system. Our formulation uses the knowledge about the system and the different actions in order to solve the observer problem in an efficient and practical way. The model incorporates different hand/object relationships and the possible errors in the manipulation actions. It also uses different control mechanisms so that the observer can keep track of the workspace of the manipulating robot. We describe the automaton model of a discrete event dynamic system in the next section and then we proceed to formulate our framework for the manipulation process. Further, we develop efficient low-level event-identification mechanisms for determining different manipulation movements in the system and for moving the observer so that the "looking" procedure is stable.

2 Discrete Event Dynamic Systems

In this section we present an overview for the development of a theory for discrete event dynamic systems (DEDS). Dynamic systems are usually modeled by finite state automata with partially observable events together with a mechanism for enabling and disabling a subset of state transitions [12,14,17,18]. We describe a recently developed framework for analyzing and controlling discrete event dynamic systems [14]. We propose that this model is a suitable framework for many vision and robotics tasks, in particular, we use the model as a high-level structuring technique for our system to observe a robot hand manipulating an object. The approach used in this framework is a state space approach that focuses on controllability issues for DEDS. We consider the issues of stability, observability and stabilizability by output feedback within this framework.

2.1 What is a discrete event dynamic system ?

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. Many existing dynamic system have a DEDS structure, manufacturing systems and communication systems are just two of them. The state space approach in representing and analyzing such systems will probably lead to more applications that might be incorporated into the framework of DEDS. It will be assumed in the development of the state space approach of analyzing DEDS that some of the events in the system are *controllable*, i.e, can be enabled or

disabled. The goal of controlling DEDS is to "guide" the behaviour of the system in a way that we consider "desirable". It is further assumed that we are able to observe only a subset of the event, i.e, we can only *see* some of the events that are occurring in the system and not all. In some cases we will be forced to make decisions regarding the state of the system and how to control a DEDS based upon our observations only.

In the next subsection we will discuss the finite state model of a DEDS. This representation of a DEDS will be used the following subsections. This model will be a simple non-deterministic finite-space automaton. Graphical representations for DEDS automatons will be used as examples to explain the different definitions and ideas to be presented. The notions of stability for a DEDS will be introduced and discussed. We then focus on the questions of observability and state reconstruction from intermittent observations of the event trajectory. Further, we address the problem of stabilization by output feedback.

2.2 Modeling

The discrete event dynamic systems under consideration can always be modeled by a non-deterministic finite-state automata with partially observable and controllable events. In particular, one can make the distinction between classical automata theory [7,10,11,19] and our representation of DEDS in terms of the state transitions. In classical automata the events are inputs to the system, whereas in DEDS the events are assumed to be generated internally by the system and the inputs to the system are the control signals that can enable or disable *some* of these events. We can represent our DEDS as 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. Some functions can also be defined on our DEDS as follows :

- $d : X \rightarrow 2^{\Sigma}$
- $e : X \rightarrow 2^{\Sigma}$
- $f : X \times \Sigma \rightarrow 2^{X}$

where d is a set-valued function that specifies the set of possible events defined at each state, e is a set-valued function that specifies the set of events that cannot be disabled at each state, and f is the set-valued function that specifies state transitions from a state under different events. An output process can be formalized simply : whenever an event in Γ happens we see it, otherwise we don't see anything.

We can visualize the concept of DEDS by an example as 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.

Also $d(1) = e(1) = \{\alpha, \delta\}$, $d(3) = \{\delta\}$, $e(3) = \phi$, $f(0,\beta) = \{0,3\}$ etc. A transition, $x \rightarrow^{\sigma} y$, consists of a source state, $x \in X$, an event, $\sigma \in d(x)$, and a destination state, $y \in f(x,\sigma)$.

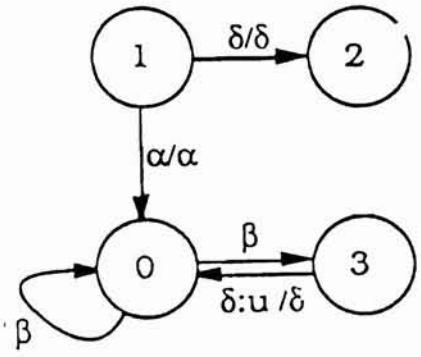

Figure 1 : A Simple DEDS Example

In general, a DEDS automaton A is a nondeterministic finite state automaton, however, if $f(x,\sigma)$ is single valued for each $x \in X$ then A can be termed as a deterministic finite state automaton. A finite string of states, $\mathbf{x} = x_0 x_1 ... x_j$ is termed a path or a state trajectory from x_0 if $x_{i+1} \in f(x_i, d(x_i))$ for *all* $i = 0...j-1$. Similarly, a finite string of events $s = \sigma_1\sigma_2...\sigma_j$ is termed an event trajectory from $x \in X$ if $\sigma_1 \in d(x)$ and $\sigma_{i+1} \in d(f(x,\sigma_1\sigma_2...\sigma i))$ for *all* i, where we extend f to Σ^* via

$$f(x,\sigma_1\sigma_2...\sigma_i) = f(f(x,\sigma_1\sigma_2...\sigma_{i-1}),\sigma_i)$$

with $f(x,\epsilon) = x$. In our graphical example (Figure 1), $\alpha\beta\beta\delta$ is an event trajectory.

If we denote a transition labeled by σ by $\rightarrow^{\sigma}$, then we can similarly let $\rightarrow^{s}$ denote a string of transitions s and $\rightarrow^{*}$ denote any number of transitions, including no transitions. We can define the range of a state x by

$$R(A,x) = \{y \in X | x \rightarrow^{*} y\}$$

indicating the set of states that can reached from x, we can also define the range of a subset of states Q in X by

$$R(A,Q) = \bigcup_{x \in Q} R(A,x)$$

An algorithm for computing $R(A,X_0)$ for any $X_0 \subset X$ that runs in $O(n)$ where $n = |X|$ can be easily formalized as follows :

Let $R_0 = Q_0 = X_0$ and iterate

$$R_{k+1} = R_k \cup f(Q_k,\Sigma)$$
$$Q_{k+1} = R_{k+1} \cap \overline{R_k}$$

Terminate when $R_{k+1} = R_k$. Then, $R(A,X_0) = R_k$.

A state $x \in X$ is *alive* if $d(y) \neq \phi$ for *all* $y \in R(A,x)$. A subset Y of X is termed a *live set* if all $x \in Y$ are alive. A system A is termed *alive* if X is a *live set.*

2.3 Stability

In this section we discuss the notions of stability and the possibility of stabilizing a discrete event dynamic system. In particular, we are going to concentrate on stability notions 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 do not stay outside for an infinite time, the problem would reduce to :

- Checking out whether all trajectories from the other states will visit E infinitely often.
- Trying to "guide" the system using the controllable events in a way such that the system will visit the "good" states infinitely often.

We shall start by defining and testing for different notions of stability and then discuss ways to stabilize a system. We shall start by assuming that the DEDS model under consideration is an uncontrolled system with perfect knowledge of the state and event trajectories ($\Sigma \cap \overline{\Gamma} = \phi$), to simplify developing the definitions and examples.

2.3.1 Pre-Stability

To capture the idea of stability , we can suppose that we have already identified a subset of states E in X that returning to E implies being in a position to continue desired behaviour from that point on. We can define the notion of a state in the DEDS being stable with respect to E in two stages. The first stage will be the weaker notion and will be termed pre-stability. We say that $x \in X$ is pre-stable if *all* paths from x can go to E in a finite number of transitions, i.e, no path from x ends up in a cycle that does not go through E.

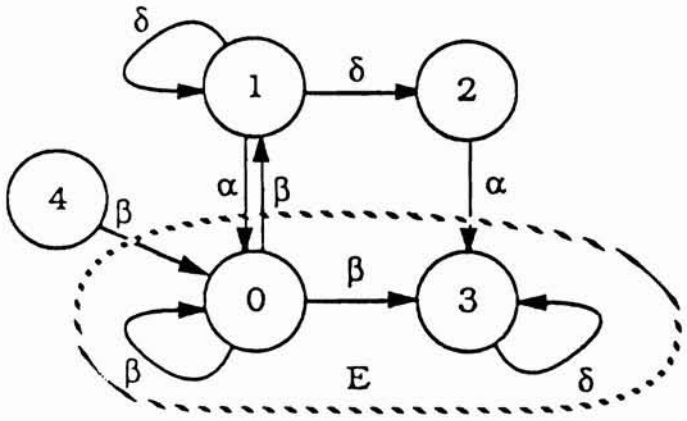

Figure 2 : Stability Example

In Figure 2, states 0, 2, 3, and 4 are pre-stable, since all transitions from them can goto {0, 3} in a finite number of transitions. State 1 is not pre-stable since it will stay forever outside E if an infinitely long string of δ's occurs. A definition of pre-stability can be formalized as follows :

Given a live system A and some $E \subset X$, a state $x \in X$ is pre-stable with respect to E (or E-pre-stable) if for all $\mathbf{x} \in \mathcal{X}(A, x)$ such that $|\mathbf{x}| \geq n$, there exists $y \in \mathbf{x}$ such that $y \in E$. We say that a set of states is E-pre-stable if all its elements are E-pre-stable and a system A is pre-stable if X is E-pre-stable.

The restriction for liveness can be flexible in the sense that if all the dead states are within E, then an automaton might still be E-pre-stable. It follows from the above definition that

a state $x \in X$ is E-pre-stable iff $x \in E$ or $f(x, d(x))$ is E-pre-stable. The following algorithm computes the maximal E-pre-stable set X_p within a system :

Let $X_0 = E$ and iterate :

$$X_{k+1} = \{x | f(x, d(x)) \subset X_k\} \cup X_k$$

Terminate when $X_{k+1} = X_k$, then $X_p = X_k$.

In Figure 2, it can be noticed that $X_1 = X_2 = X_p = \{0, 2, 3, 4\}$.

2.3.2 Stability

The stronger notion of stability corresponds to returning to the set of "good" states E in a finite number of transitions following any excursion outside of E. Thus, given E, we define a state $x \in X$ to be E-stable if all paths go through E in a finite number of transitions and then visit E infinitely often. As an example, in Figure 2, where $E = \{0, 3\}$, only 2 and 3 are stable states. State 1 is not stable since the system can loop at 1 infinitely. State 0 although in E is not stable since the system can make a transition to 1 and then stays there forever, the same applies to state 4. We can use the previously defined notion of pre-stability and define a state to be E-stable if all the states in its reach are E-pre-stable. In Figure 2, 0 and 4 are not E-stable since they can reach 1, which is not E-pre-stable. We can define stability as follows :

Given a live A and $x \in X$, x is E-stable iff $R(A, x)$ is E-pre-stable. A $Q \subset X$ is stable if all $x \in Q$ are stable. A system A is stable if X is a stable set, from which we can conjecture that A is E-stable iff it is also E-pre-stable.

2.3.3 Pre-Stabilizability

Now, we introduce control and reconsider the stability notions discussed before. We try to "guide" our system or some states of it to behave in a way that we consider desirable. Pre-stabilizability is described as finding a state feedback such that the closed loop system is pre-stable. We can then define pre-stabilizability formally as follows :

Given a live system A and some $E \subset X$, $x \in X$ is pre-stabilizable with respect to E (or E-pre-stabilizable) if there exists a state feedback K such that x is alive and E-pre-stable in A_K. A set of states, Q, is a pre-stabilizable set if there exists a feedback law $K(s)$ (A control pattern) so that every $x \in Q$ is alive and pre-stable in A_K, and A is a pre-stabilizable system if X is a pre-stabilizable set.

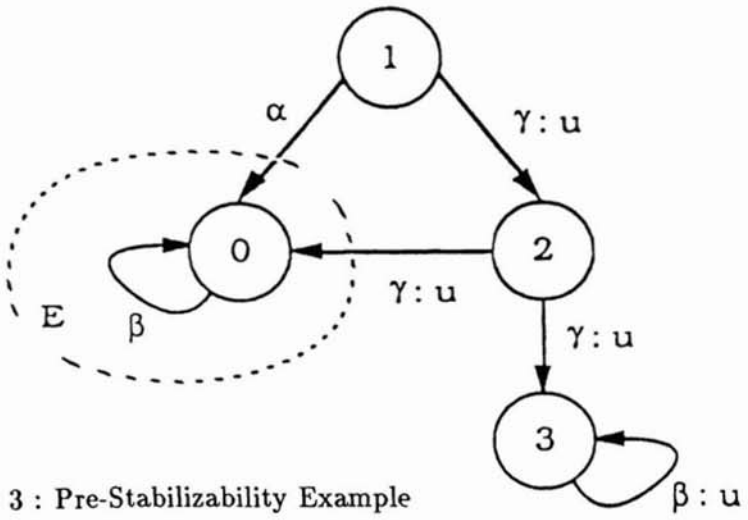

Figure 3 : Pre-Stabilizability Example

As an example, in Figure 3, state 1 is pre-stabilizable since disabling γ pre-stabilizes 1. However, disabling γ at state 2 leaves no other defined events at 2 and "kills" it, so neither state 2 or 3 is pre-stabilizable.

2.3.4 Stabilizability

Stabilizability is an extension of pre-stabilizability. Stabilizability is described as finding a state feedback such that the closed loop system is stable. We can then define stabilizability formally 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 state feedback K such that x is alive and E-stable in A_K. A set of states, Q, is a stabilizable set if there exists a feedback law $K(s)$ (a control pattern) so that every $x \in Q$ is alive and stable in A_K, and A is a stabilizable system if X is a stabilizable set.

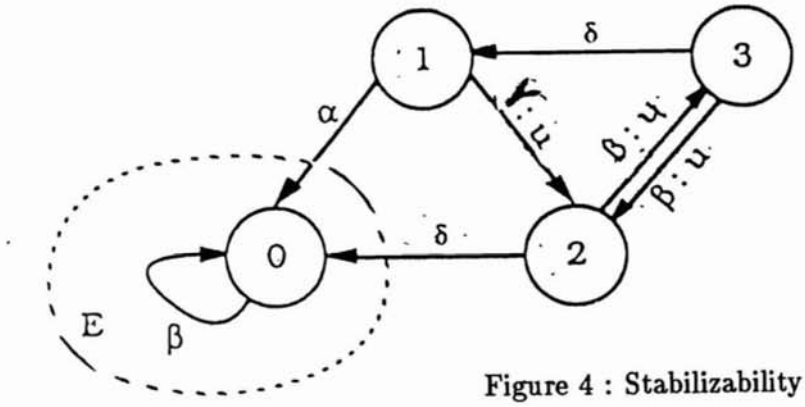

Figure 4 : Stabilizability Example

In Figure 4, disabling β at state 2 is sufficient to make the whole system stable with respect to state 0. Disabling γ at state 1 will help stabilize only state 1, because the system can then continue looping between states 2 and 3. Disabling β at state 3 will not help stabilize or pre-stabilize any state.

2.4 Observability

In this section we address the problem of determining the current state of the system. In particular, we are interested in observing a certain sequence of *observable* events and making a decision regarding the state that the DEDS automaton A might possible be in. In our definition of observability, we visualize an intermittent observation model, no direct measurements of the state are made, the events we observe are only those that are in $\Gamma \subset \Sigma$, we will not observe events in $\Sigma \cap \overline{\Gamma}$ and will not even know that any of which has occurred. State ambiguities are allowed to develop (which must happen if $\Sigma \neq \Gamma$) but they are required to be resolvable after a *bounded* interval of events. This notion of observability can be illustrated graphically as in Figure 5.

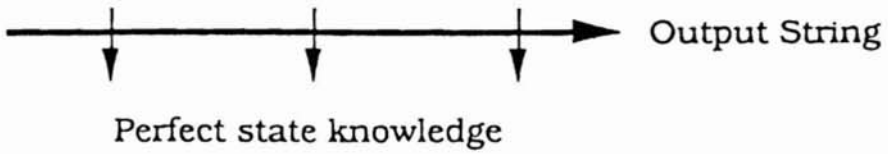

Figure 5 : Notion of Observability

2.4.1 Requirements

In developing the theory and examples we shall concentrate on uncontrolled models of DEDS automatons with partial knowledge of the event trajectory. Due to the fact that we are "seeing" only observable events in Γ in our system, it is not desirable to have our automaton generate arbitrarily long sequences of unobservable events in $\Sigma \cap \overline{\Gamma}$. A necessary condition to guarantee this is that the automaton after removing the observable events $A|\overline{\Gamma}$, must not be alive. In fact, it is also essential that every trajectory in $A|\overline{\Gamma}$ is killed in finite time by being forced into a dead state. It can be seen that the condition for a DEDS automaton to be unable to generate arbitrarily long sequences of unobservable events, is that $A|\overline{\Gamma}$ must be D-stable, where D is the set of states that only have observable events defined (i.e, $D = \{x \in X | d(x) \cap \overline{\Gamma}\}$).

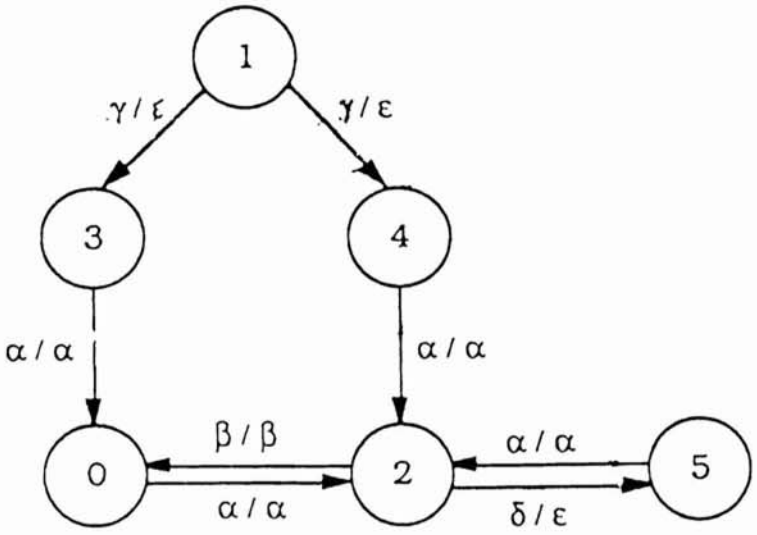

Figure 6.1 : A Simple System

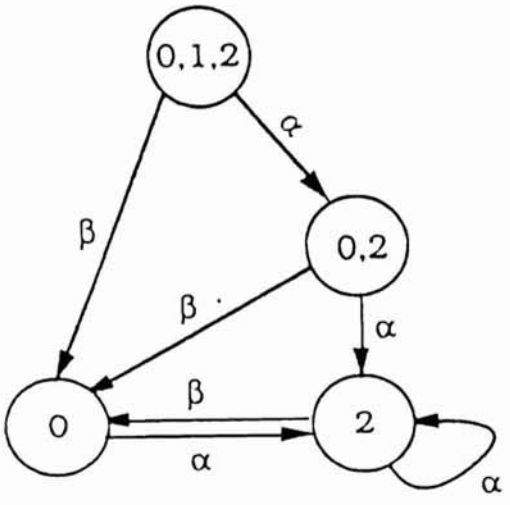

Figure 6.2 : Observer for the System in Figure 6.1

2.4.2 State Observability

As illustrated in Figure 5, a DEDS is termed *observable* if we can use the observation sequence to determine the current state exactly at intermittent points in time separated by a *bounded* number of events. More formally, taking 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.

In Figures 6.1 and 6.2 a simple system and its observer are illustrated. It can be seen that the observer will never know when will the system be in states 3, 4 or 5, since the events that takes the system to those states are unobservable (δ/ϵ means that $\delta \in \Sigma \cap \overline{\Gamma}$), namely δ and γ. There are two states in the observer which are ambiguous, however, another two states are singleton states, i.e, when our observer reaches them, we'll know the exact state that the DEDS in currently in. Had it been the case that our observer could, for example, loop forever in ambiguous states, then the DEDS would be unobservable. This leads to the following formal definition of observability that ties it with the notion of stability :

A DEDS automaton A is observable iff E is nonempty and O is E-stable.

where O is the observer for A and E is the set of singleton states of O. It can be seen that the observer in Figure 6.2 is stable with respect to the nonempty subset of states {0, 2} and thus the DEDS of Figure 6.1 is observable.

2.5 Output Feedback Stabilizability

In this section we combine the ideas discussed in the previous two subsections regarding observability and stability to address the problem of stabilization by dynamic output feedback under partial observations. In this section we concentrate on partially controlled systems with partial knowledge of the event trajectory. In particular, our goal is to develop stabilizing compensators by cascading and a stabilizing state feedback defined on the observer's state space.

2.5.1 Requirements

To attack the problem of output feedback stabilization, it should be noticed that we are actually trying to "manipulate" the system's observer, in other words, what we have available in a sequence of observable events (the system's *output*) and we are trying to use this output to control the behaviour of the system using ***only*** the events that we can control. It is then possible to redefine the problem of output feedback stabilization as the stabilization of the observer by state feedback.

The obvious notion of output E-stabilizability (stabilizability with respect to $E \subset X$) is the existence of a compensator C so that the closed-loop system A_C is E-stable. It is possible that such a stabilizing compensator exists, such that we are sure that the system passes through the subset E infinitely often (E-stable) ***but*** we never know when the system is in E. A stronger notion of output feedback stabilizability would not only requires that the system passes through subset E infinitely often, but also that we regularly know when the system is in E. In out example and discussion we shall concentrate on this stronger notion of output stabilizability.

2.5.2 Strong Output Stabilizability

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 fact that the observer state should be a subset of E instead of having the observer state of interest *includes* states in E is because we want to *guarantee* that our system in within E. Our compensator should then *force* the observer to a state corresponding to a subset of E at intervals of at most a finite integer i observable transitions. We can then formalize the notion of a strongly output stabilizable system as follows :

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 $E_O = \{ \hat{x} \in Z \mid \hat{x} \subset E \}$.

where Z is the set of states of the observer.

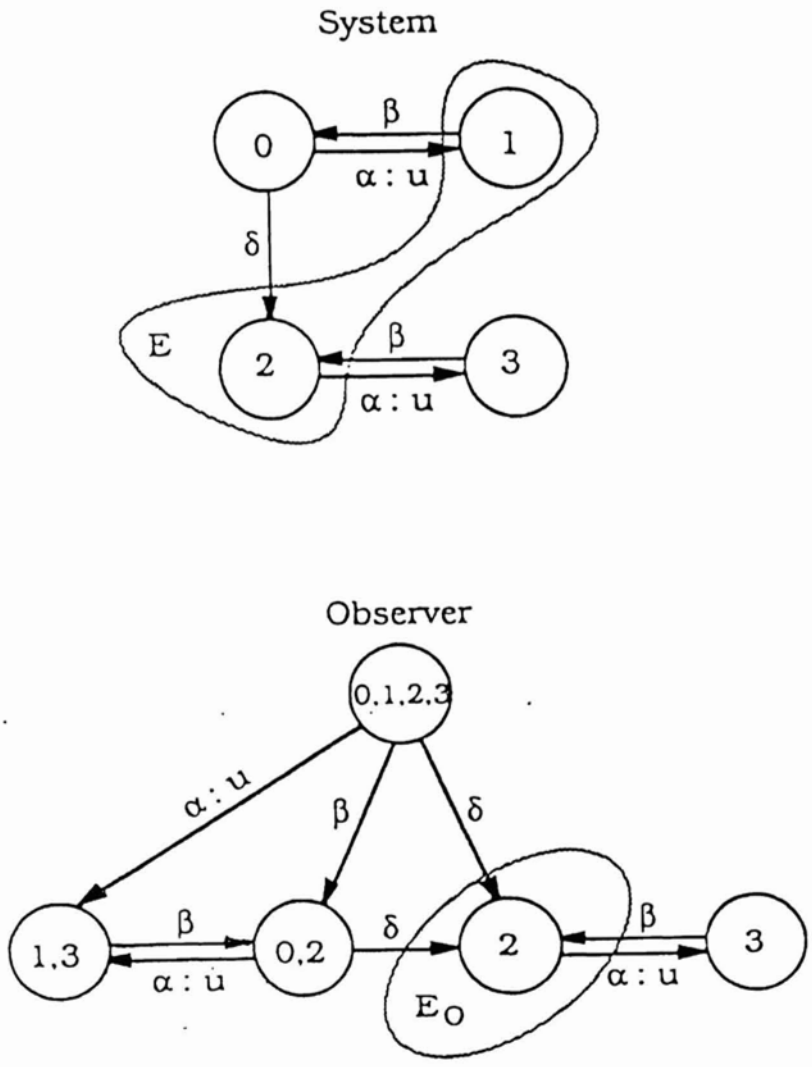

Figure 7 : Example for Output Stabilizability

As an example, considering the DEDS and its observer in Figure 7, where $E = \{1, 2\}$, we have to check the observer stability (or stabilize the observer) with respect to E_O, because this is the only observer state that is a subset of E. As a start, we do not know which state is our system in (as denoted by the state $\{0, 1, 2, 3\}$), however, using the observer transitions we can see that to achieve E_O-stability for the observer we only need to disable α at the observer state $\{0, 2\}$. It should be noted that all the events are observable in this DEDS automaton.

3 Modeling and Observer Construction

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. We avoid the excessive use of nested decision structures and exhaustive searches when observing the 3-D world motion and structure.

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, shaking, grasping, 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.

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 much more complicated models for more tasks can be built. More views for the hand can be allowed and more events. The example shown here is for illustration purposes only.

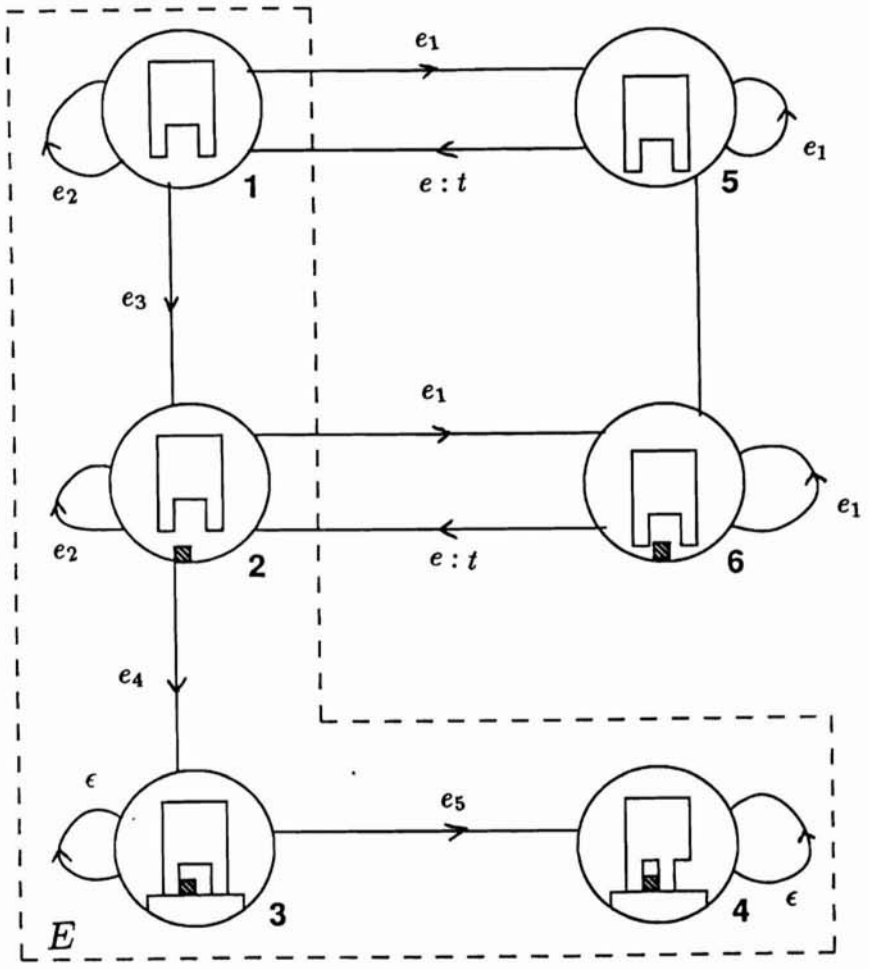

Figure 8 : A Model for a Grasping Task

As shown in Figure 8, the model represents a frontal 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. 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. The state 5 is the state representing the appearance of the empty hand after having moved downward, the hand is not in a good visual position with respect to the viewer as it tends to escape the camera imaging plane. The same applies to state 6, the only difference is that the object is in sight. States 5 and 6 are considered as bad states, as the system will tend to go into a non visual states 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 that causes state transitions and as the appearance of the object in the camera image plane. 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 the downward movement of the hand. The self loops are caused by the stationarity of the scene with respect to the viewer, that is, no movement of any kind has happened. In the next section we discuss 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 downward movement of the hand. Tracking techniques will be discussed in details in the next section. 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 events.

It should be mentioned that this model of a grasping task could be extended to allow for error detections 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 to determine the current state of the system 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 observer. Disabling the tracking events will obviously make the system neither stable nor 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. The whole system is stabilizable w.r.t. E, enabling the tracking event will cause all the paths from all the states to go through E in a finite number of transitions and then visit E infinitely often.

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, it can be easily seen that, for the model we have developed for the grasping task, the system can be stable with respect to the set E_O as defined in the previous section. An observer can be formed for the system as shown in Figure 9.

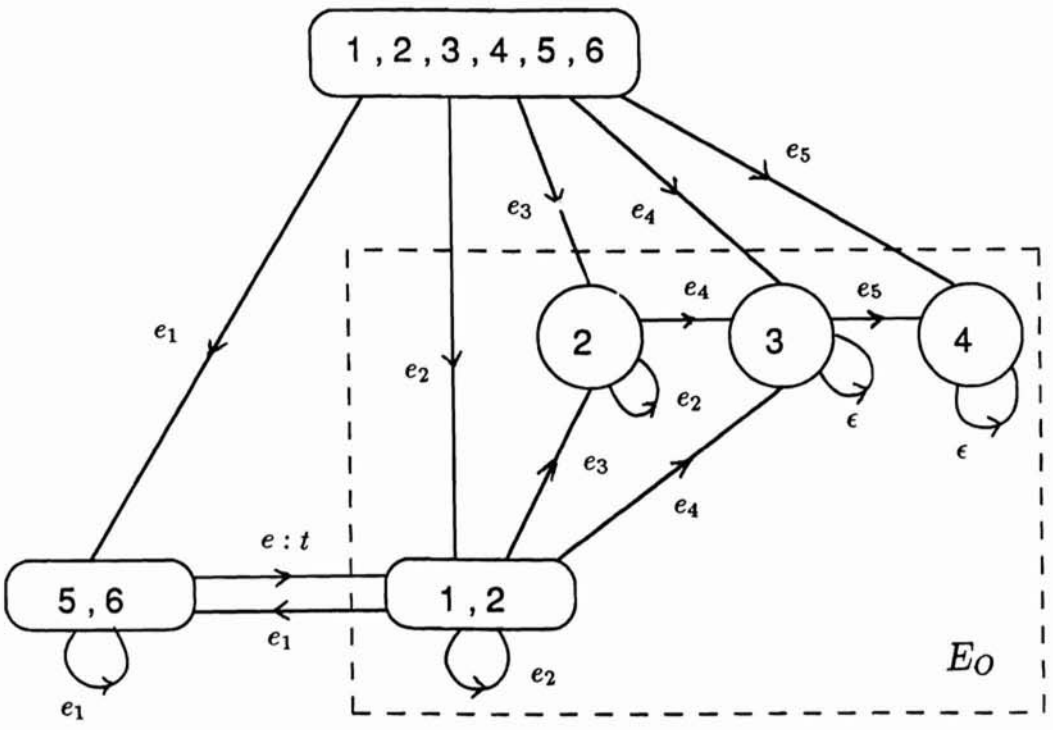

Figure 9 : Observer for the Grasping System

At the start, the state of the system is totally ambiguous, however, it is obvious that the observer can be "guided" to the set E_O consisting of all the subsets of the good states E as defined on the 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 be made stable w.r.t. E_O and thus the system is strongly output stabilizable. The singleton states represents the instances in time where the observer will be able to determine without ambiguity the current state of the system. Next, we shall discuss mechanisms for recognizing the different events that causes state transitions.

4 Event and State 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 section. We also describe a simple mechanism for identifying a visual state for the system. We start by identifying the manipulating hand and the object (if it exists) within the 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.

4.1 Image Motion of the Hand

In order to be able to identify how the manipulating hand is moving within a grasping task, we use the image motion to estimate the hand movement. Many techniques were developed to estimate the optic flow (the 2-D image motion vectors) [2,6,9,16,27], 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 moving hand.

As a start, we use a simple two-dimensional segmentation scheme in order to identify the hand and the manipulated object within the camera view. The input image is thresholded, and all the "objects" within an image are identified. An object is simply characterized by a region with a space of at least one pixel surrounding it from every where, thus regions with holes can be easily recognized using this technique. We can assume that the largest object in the figure is the hand and the second largest object is the manipulated object, or we can make our decision built on the knowledge we have regarding the geometry of the hand and the object.

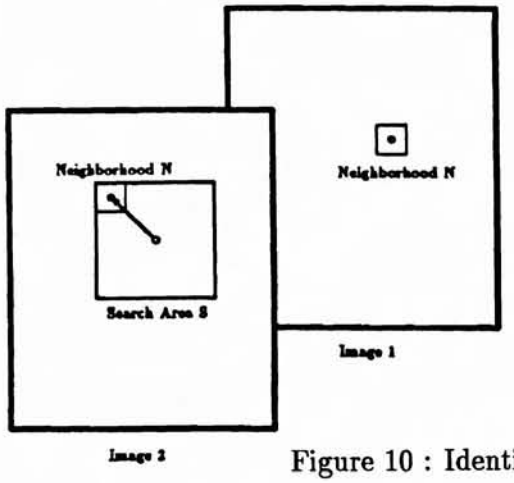

Figure 10 : Identifying the SSD Optical Flow

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 10. 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 $(1, 1)$. Figures 11.1 and 11.2 illustrates the aperture problem.

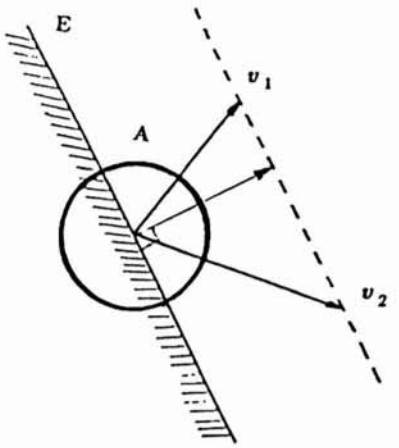

Figure 11.1 : The Aperture Problem
The direction of motion of edge E cannot be determined by viewing E through the aperture A

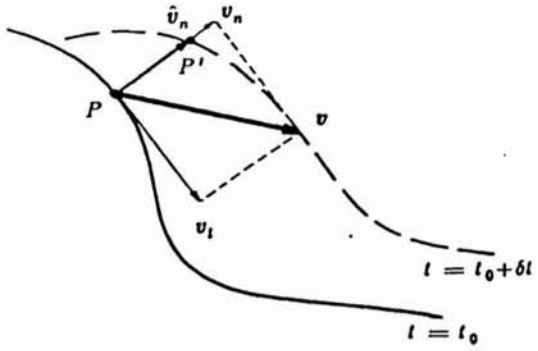

Figure 11.2 : Normal Flow Estimation

The third problem is that the scheme will only determine the displacement to pixel accuracy. 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. 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, as shall be explained later.

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 12. 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.

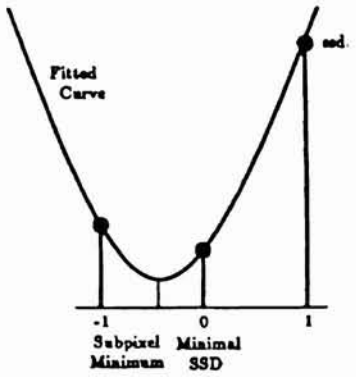

Figure 12 : Subpixel Accuracy for Optical Flow

A simpler version of the above algorithm can be implemented in real-time using a multi-resolution approach [27]. We can 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.

Knowing that we are in a specific state of the system model of a grasping task, where a downward movement of the hand is expected, for example, then we can use the flow detection mechanism to check whether most of the flow vectors point down wards, that is, the normal flow vector points downward with a small least square error of fit. If that is the case, then the event is identified with a reasonable accuracy. We can supplement our flow algorithm with a simple module to calculate the zeroth and first order moments of the hand (and object, if any is found) to check how the center of mass and area of the image "objects" have varied from the previous image, if the variation of area is negligible and the center of mass have varied in a consistent manner with the identified full flow estimation then the corresponding tracking action is simple to perform as shall be described later.

4.2 Tracking and State Recognition

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 manipulation action might be a simple one that does not require moving the camera at all, 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.

When the recovered full hand flow on the 2-D image plane of the camera is a simple downward movement of the hand (in a grasping task) and the center of mass movement substantiates the claim that the hand is approaching the object, then, if the variation of the hand 2-D area on the image plane is negligible, the image flow in pixels can be transformed to physical motion on the CCD array of the camera, given such parameters as the focal length of the camera and the CCD array width and length in cm. We can then use the transformation between the camera CCD array coordinates and the robot end-effector coordinates to calculate in 3-D coordinates the corresponding cartesian motion of the robot arm to compensate for the hand movement on the CCD array of the camera, and thus "track" the hand. It might be the case that the hand area has changed significantly. This implies that the hand has approached or has moved away from the observer, in that case, depth information would have to be extracted and used when we move the observer robot. A subset of the three-dimensional motion and structure parameters would have to be calculated using two or more frames [4,22,24,26]. 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.

Two kinds of tracking can be used, in the first kind, the two images for 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 take photos 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.

One can model an arbitrary 3-D motion in terms of stationary-scene/moving-viewer as shown in Figure 13. The optical 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 [13] :

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

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

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.

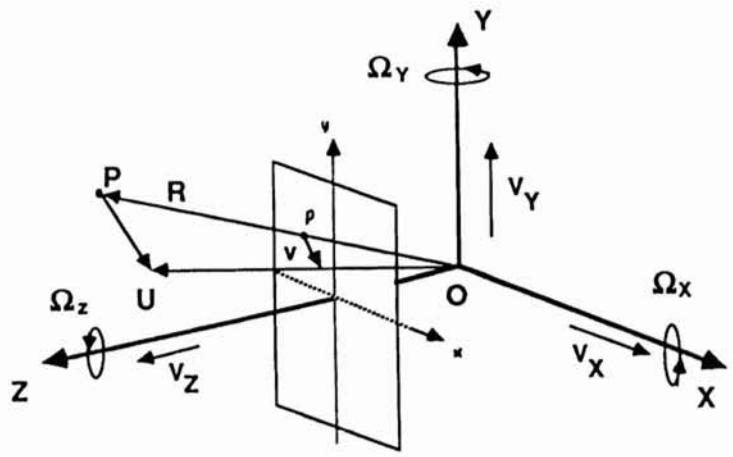

Figure 13 : 3-D Formulation for Stationary-Scene/Moving-Viewer

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 the absence of translations, we can compensate for the rotational part in a very fast and efficient way. Using only rotations for tracking, one can solve for the image flow of the hand, after doing the shift corresponding to the image flow due to the observer's motion.

As a part of the events definition, as mentioned in the previous section, is the identification of the orientation of the hand and/or the object relative to the observer's camera. The hand might be in an upright position or a tilted one, different views of the hand might be seen by the observer as the manipulation process evolves over time (for example, elevation, plan or side views). In a situation like a simple grasping task, as defined in the previous section, one might need only to check for a specific upright configuration of the hand in most of the states of the system observer's states. Depending on the current state of the observer, and on the different possible configurations of hand as defined by the system and the allowed manipulation actions, checking for a particular orientation, or for one of a number of possible orientations (at a particular observer state) is performed. The appearance of the object can be checked by analyzing the output of the simple 2-D segmentation algorithm that we described before.

Different techniques can be used for asserting the way the hand "looks" to the observer's camera. Depending on the system states and the allowed manipulation processes, we can know a-priori the possible configurations of the hand. One way might be to *parameterize* the hand, that is, try to identify different ratios, for example, between the length, width and the space between the claws, to distinguish between different views. Extensive computations and searches are the drawbacks of this strategy. Another approach can be to binarize the hand image, after 2-D segmentation and performing edge tracing, then we can compare the closed, one pixel wide contour resulting from the operation with pre-stored, scaled contours, corresponding to "ideal" orientations of the hand image. The contour that matches best is the one with the least squared error of match, however, this method is very susceptible to noise, which was discovered experimentally. A simple way to determine the hand configuration is to store some binarized, ideal orientations of the hand with a small number of pixels for

each orientation, typically 64 x 64 pixels. We then compare all the pixel values in the small windows to their corresponding locations (after scaling) in the segmented and binarized observer's image, the number of mismatches is recorded and the window with the least number of mismatches is chosen to be the current configuration of the hand.

5 Results

The Lord experimental gripper is used to perform a grasping task as defined by our model. Different views of the gripper are shown in Figures 14.a to 14.c. The hand approaches an object and then grasps it. The sequence of visual states as seen by the observer camera is shown in Figures 15.a to 15.d. We use the simple version of the image flow detection algorithm to compute the optic flow of the gripper movement, the flow vectors resulting from applying the algorithm to diagonal movements of the gripper are shown in Figures 16.a and 1b.b, the movements are upwards to the left and downwards to the right. It can be seen that the image flow vectors are consistent with the actual motion.

The event identification mechanisms that were described above are used for recognizing the hand motion and the configuration of the current state. The image gaussian pyramid for the gripper is shown in Figure 17, the pyramid is formed by the successive application of gaussian low-pass filtering and decimation by half, five levels of the pyramid are shown. Edge tra' ing results for the output of the segmentation module are shown in Figure 18. The result of the two-dimensional segmentation mechanism for identifying the hand orientation is shown in Figure 19. The silhouette is used to recognize the hand current configuration.

6 Conclusions and Future Research

We have proposed a new approach to solving the problem of observing a manipulation process. Our approach uses the formulation of discrete event dynamic systems as a high-level model for the framework of evolution of the hand/object 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 observer robot performs different tracking events in order to guide the process towards stability and to keep track of the three-dimensional workspace of the manipulating robot. We have described algorithms for identifying the events that causes state transitions within the manipulation process and for state identification.

The proposed system 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. 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, pushdown automata, turing machines and/or μ-recursive functions. 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.

References

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

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

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

[4] 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.

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

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

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

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

[9] 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.

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

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

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

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

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

[15] 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.*

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

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

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

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

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

[21] 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.

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

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

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

[25] 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.

[26] 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.

[27] 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.

Figure 14.a

Figure 14.b

Figure 14.c

Different Views of the Lord Gripper

Figure 15.a

Figure 15.b

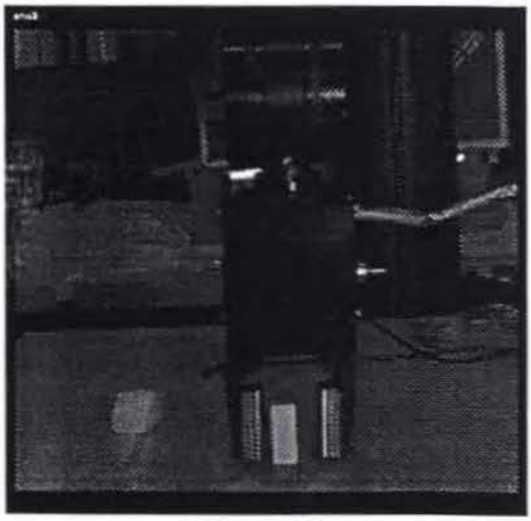

Figure 15.c

Figure 15.d

The Grasping Task

Figure 16.a

Figure 16.b

Optic Flow Vectors

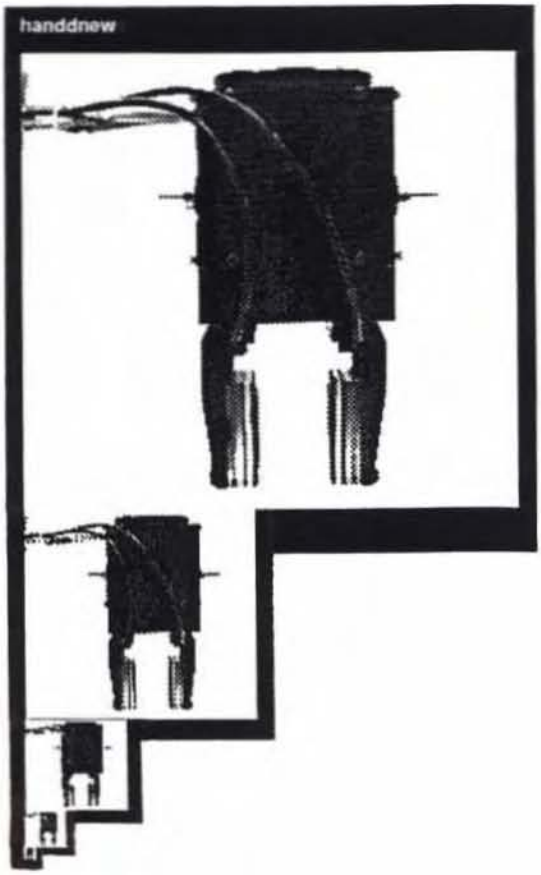

Figure 17 : Gaussian Pyramid of the Hand

Figure 18 : Results of Edge Tracing

Figure 19
Results of 2-D Segmentation to Find the Hand