ELSEVIER

Robotics and Autonomous Systems 12 (1994) 187–198

Robotics and Autonomous Systems

Discrete Event Systems for autonomous mobile agents

Jana Košecká, Ruzena Bajcsy *

GRASP Laboratory, Dept. of Computer and Information Science, University of Pennsylvania, 3401 Walnut Street, 301 C, Philadelphia, PA 19104, USA

Abstract

Discrete Event Systems (DES) are a special type of dynamic system. The 'state' of these systems changes at discrete instants in time and the term 'event' represents the occurrence of discontinuous change (at possibly unknown intervals). Different Discrete Event Systems models are currently used for specification, verification, synthesis as well as for analysis and evaluation of different qualitative and quantitative properties of existing physical systems. The focus of this paper is the presentation of the automata and formal language model for DES introduced by Ramadge and Wonham and its application to the domain of mobile manipulator/observer agents. We demonstrate the feasibility of the DES framework for modeling, analysis and synthesis of some visually guided behaviors of agents engaged in navigational tasks and address synchronization issues between different components of the system. The use of DES formalism allows us to synthesize complex behaviors in a systematic fashion and guarantee their controllability.

Key words: Visually guided behaviors; Control; Reactive behaviors; Discrete Event Systems; Obstacle detection; Obstacle avoidance; Cooperation; Real-time vision; Mobile robots

1. Introduction and motivation

There has been a long tradition in the AI and robotics community to incorporate reactive and behavior-based components into the design of autonomous systems. Among the early approaches were Arkin's schemas [2] motivated by neuroscientific sources, demonstrating how different perception/action schemas should be instantiated based on the task to be accomplished.

Some of the ideas on how to compose different behaviors have culminated in the subsumption architecture [5]. In this approach simple behaviors are encoded and combined in a subsumption like manner and more complicated behaviors emerge from them. Another group of researches believe that purely reactive approaches won't be sufficient to accomplish more complex goals and are trying to incorporate reactive approaches into traditional AI planning systems [16,7].

The reactivity of the above-mentioned systems is achieved by tight coupling of sensors and actuators. The sensors used for these purposes were mostly sonar, infrared sensors or simple vision

* Correspondence can be sent to either author. E-mail: bajcsy@grip.cis.upenn.edu *or* janka@grip.cis.upenn.edu

SSDI 0921-8890(93)E0043-M

based sensors [9], where sensory readings directly, or after some simple computation, determine appropriate commands for actuators.

In the case of more complex sensory guided behaviors (vision for navigation/observation) this coupling is no longer straightforward. This is mostly due to the fact that the amount and the type of visual information needed is task dependent and moreover varies substantially even within the task. In navigation for instance one would like to locate landmarks and also avoid obstacles, while in exploration one would like to recover the 3D structure of the environment in an active way. In these situations it is not clear how to 'hard-wire' the appropriate connections between sensors and actuators in order to obtain the desired behavior.

In this paper we embed the problem of modeling tasks and reactive behaviors of autonomous agents into the DES framework developed by Ramadge and Wonham [13]. The DES formalism models systems in terms of finite state automata and allows *observations* (the qualitative information extracted from sensory data) and *actions* (commands to the actuators of the vehicle) to be treated in a uniform way in terms of *events*. Events correspond to the transitions between states and model (abrupt) discrete changes in the system's behavior. The DES framework is suitable for investigating some control-theoretic properties of the system such as *controllability* and *observability* which can be conveniently predicted. Moreover, different visually guided behaviors can be combined in a modular fashion such that the resulting behavior will be guaranteed to be controllable. Within this framework one can also address the combination of different models in a hierarchical fashion.

The task

We are engaged in an effort of investigating cooperation of two mobile robots whose task is to navigate in a cooperative fashion. The two mobile robots are equipped with visual capabilities and some additional sensors (see Fig. 1) and their task is to follow a path individually or together, while keeping a particular formation and avoiding unexpected obstacles. This task implies the following subtasks:

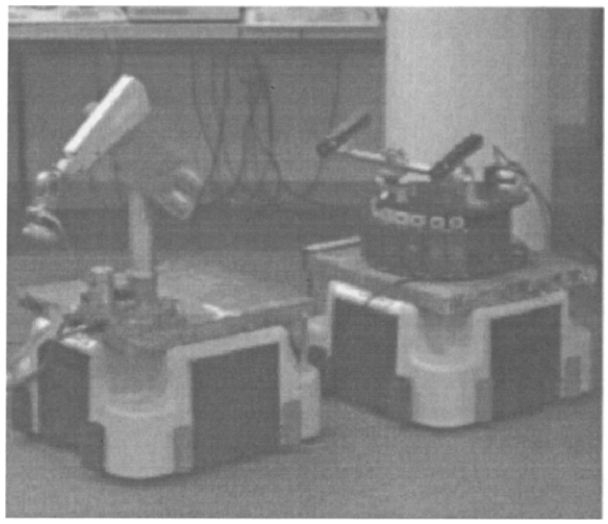


Fig. 1. Two mobile agents.

1. development of real-time vision processing capabilities that enable visually guided navigation, both in free space as well as in the presence of obstacles,
2. development of recognition and tracking capabilities of the other mobile robot,
3. development of a control structure for each mobile robot that couples the results of visual processing with the control,
4. development of the control architecture that will enable the cooperative behavior of the two mobile robots,
5. representation of the visually extracted information from the surrounding environment that is necessary to assure cooperative behavior.

Overview

In Section 3 we introduce the supervisory control theory of DES and its extension relevant to modeling behaviors of mobile agents. We then describe vision algorithms for obstacle detection, avoidance and tracking, emphasizing the qualitative information extracted. In Section 4 we formulate navigation and tracking as control problems and propose the DES models of these visually guided behaviors. The feasibility studies have been performed in the laboratory environment

with two TRC vehicles, under different illumination conditions and unexpected obstacles.

2. Discrete event systems

Discrete Event Systems are mostly man-made systems arising in the domains of manufacturing, robotics, organization and delivery services, vehicular traffic, and computer and communication networks. The behavior of these systems is truly *nonlinear* and they have *time-varying parameters*. Due to the nonterminating interaction with the environment these systems are often affected by unexpected interventions, which means that they are *discontinuous*. Moreover, they often demonstrate *uncertain behavior*, caused by the fact that available measurements and inputs are often disturbed by noise. Therefore, the evolution of the system may be unpredictable (nondeterministic). The interactions between particular components are complex and no longer governed by known physical laws describable by differential equations. Due to the complexity of these systems, there is a need to model and analyze them at different *hierarchical* levels. Because of the special nature of these systems, in the past different formal methods were proposed for their modeling, emphasizing different aspects of the system design and analysis. The main objective of these efforts was to assure the appropriate behavior of the system, as well as its full functionality in the given environment, by means of appropriate control. One class of models used successfully to study different qualitative properties of DES are so-called *logical models*. The behavior of the system using logical models is expressed in terms of system trajectories, i.e. listings of events that occur along the sample path. Typically the set of possible trajectories is specified first. This can be done using some form of transition structure (automata, Petri nets) or by means of algebraic equations (CSP, finitely recursive processes), or by logical calculus (temporal logic). Further, some constraints can be imposed on the system specifying its desired behavior (expressed in terms of all admissible trajectories). The trajectories are then investigated whether they satisfy the desired properties expressed by the constraints. Properties of interest may include stability, convergence, correct use of resources, correct event ordering, deadlock, liveness etc. The need for examining these different qualitative properties has resulted in the development of a large variety of tools employed in such areas as semantics of concurrent programs, communicating sequential processes, synchronization issues in operating systems, communication protocols in networking and logical analysis of digital circuits. Logical models may be used as a basis of verification and synthesis of DES. An overview of different approaches can be found in [13].

3. Supervisory control theory

The following section is a brief introduction to *supervisory control theory* of Discrete Event Systems [13]. DES are a class of dynamic systems which evolve with the abrupt asynchronous occurrence of physical events at possibly unknown intervals. In contrast to the familiar class of dynamic systems, in which the physical world is described by differential equations and state trajectory is continuous, in DES state changes asynchronously at discrete instances of time. The DES are modeled as nondeterministic finite state machines, where states correspond to some continua in the task evolution and the transitions between them are caused by events, representing the qualitative changes in environment or task evolution. Let Σ denote the set of events that the system can generate. Then event trajectories can be thought of as strings over this fixed alphabet Σ. Let the subset $L \subseteq \Sigma^*$ represent all event trajectories which are physically possible for the system and fully characterize its behavior. This *language* L can be described in terms of a machine which generates all strings from L.

We define a generator $\mathcal{G}$ to be a 5-tuple

$$\mathcal{G} = (Q, \Sigma, \delta, q_0, Q_m),$$

where

Q is the set of all possible states,
Σ is the set of all possible event,
δ is the transition function $\delta : \Sigma \times Q \to Q$,

q_0 is the initial state,
Q_m is the subset of states called marker state, $Q_m \subseteq Q$.

In order to adjoin means of control to the system, events are classified into two categories: *uncontrollable events* which can be observed, but cannot be prevented from occurring (e.g. obstacle detected, robot failure) and *controllable events* that can be prevented from occurring (e.g. stop or move). In the original framework the model consists of the *'plant'* – the subject of control embedded in the environment which generates events. The goal of the control is to enable or disable controllable events in order to ensure correct behavior of the plant.

Since the open-loop behavior of the system is often unsatisfactory, the next main objective is to design a controller for a plant in such a way that the plant behaves in accordance to the specified constraints. Such a controller is called a *supervisor*. A supervisor can be thought of as the map

$$f: L \to \Gamma,$$

where $\Gamma: \{0, 1\}^{\Sigma_c}$ is the set of all binary assignments to the elements of Σ_c. These binary assignments determine the control pattern for each state (i.e. which events should be enabled and which should be disabled.)

Function f then specifies for each possible string from the language L the control input $f(w)$ in Γ to be applied at that point. This mapping can be also realized by a pair

$$\mathcal{S} = (S, \phi),$$

where S is an automaton

$$S = (X, \Sigma, \xi, X_0, X_m),$$

and

$$\phi: X \to 2^{\Sigma} \text{ s.t. for each } w \in L(\mathcal{S}),$$

$$f(w) = \phi(\xi(w, x_0)),$$

Thus, a clear distinction is made between the subject of control, the *plant*, and the agent doing the controlling, the *supervisor*. The existence of a supervisor for a given plant, i.e. the existence of an appropriate feedback control, is very closely related to the concept of *controllability*. A system is said to be *controllable* when based on the information of the current state of the system and by means of appropriate control, we can reach any desired state of the system. Let language $L \subseteq \Sigma^*$ represent all possible trajectories. We say that the *language K is controllable* iff

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K},$$

where K represents the desired behavior of the system and $\overline{K}$ is the prefix closure of K. In other words, this condition requires that for any prefix of a string in K, i.e. any $w \in \overline{K}$, if w is followed by an uncontrolled event $\sigma \in \Sigma_u$ in L, then it must also be a prefix of K. In a more intuitive way, since the uncontrollable events cannot be prevented from occurring, it is clear that if such an event occurs, then the path along which that event occurred must remain in K in order for K to be controllable (feasible closed-loop behavior). If the plant's desirable behavior is characterized by a language K, and K is controllable and prefix

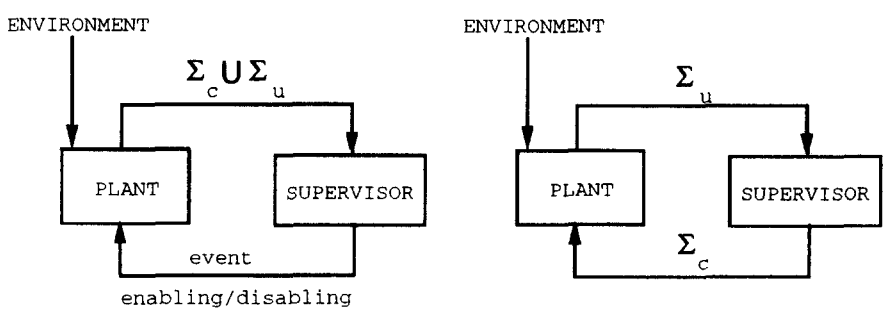


Fig. 2. Generator vs. I/O semantics.

closed, then the existence of a supervisor is guaranteed [14].

3.1. Modular composition

The system under investigation is usually composed from the modular subsystems. When each of the subsystems (plants) is modeled by the process P_i the global plant is obtained as $P = \| P_i$. The symbol $\|$ is the composition operator (shuffle product) which models asynchronous composition, where the events of each of the subsystems can occur independently from the others. This type of composition models the *interleaving model* of concurrency often used in theory of concurrent systems.

3.2. Input / output semantics

There are different ways to interpret the communication between the plant and the supervisor. In the original framework the plant 'generates' events and the supervisor enables or disables controllable events (see Fig. 2a). Only the plant is responsible for scheduling an event occurrence. Sometimes this type of semantics might not be appropriate. There are often situations where not all events in the plant occur spontaneously. Some of them may be triggered (forced to occur) by the supervisor or occur as responses to the commands previously triggered by the supervisor. The set of events Σ is therefore partitioned to Σ_u – responses and Σ_c – commands. This I/O semantics of the original model was introduced by [3] and is depicted on Fig. 2b. For the examples shown in this article we have adopted generator sematics of DES. We will discuss implications of this choice in Section 4.4.

3.3. Closed-loop system behavior and synchronization

In order to achieve the desired closed-loop system behavior the issues of synchronization between the plant and the supervisor must be addressed. In the original framework the plant and the supervisor are *fully synchronized*, i.e. every time an event occurs in the plant and a transition to another state is made, the transition on the same event is made in supervisor. This model of synchronization is not always desirable. We can envision situations when the plant makes transitions due to some uncontrollable or unobservable events and the supervisor cannot interfere with their execution. In such cases it may not be necessary for the supervisor to 'track' these events (i.e. to make the transition synchronously with the plant every time such an event occurs). Another situation may occur when it is not appropriate for the plant to execute commands suggested by the supervisor. For example, when a supervisor requests an action which is not executed by the plant, failure of the plant can be discovered. Also in the case when a single supervisor controls more than one plant, we may envision situations when some of the commands are applicable only to one of the plants and should be ignored by the others (e.g. a single supervisor controlling both actuators of a mobile platform and camera head). In situations like these the less restrictive model of synchronization called *prioritized synchronization* (PSC) can be adopted. In this model priority sets of events are associated with each of the processes, determining the events to which processes are responsive [8,17]. [1]

4. Visually guided navigation and tracking as control problems

In this section we formulate navigation and tracking as control problems and identify particular components of the system needed for adopting the DES framework introduced in the previous section.

The subject of control is the mobile robot in an unknown or partially known environment, being committed to follow some trajectory. In order to achieve the desired goal we need to provide appropriate commands to the motors of the vehicle preventing crashes or losing the tracked tar-

[1] An event is executable in the PSC of two systems if it is executable in the system whose priority set contains that event.

get. The traditional control theory which describes the relationship between input, state and output variables of the system via differential equations is not sufficient. This is because the mobile robot must take into account unexpected interactions with the environment, and adjust its behavior to them. For example, a simple unexpected occurrence of an obstacle, or a sudden appearance or loss of the target (i.e. vehicle in front) represents a discrete change in the evolution of the system's behavior.

We describe two particular modules (navigation and tracking) in terms of *events* representing discrete qualitative changes in the system's behavior. For the purposes of accurately modeling the behavior of our system we classify the set Σ of events as follows:

- A set of uncontrollable events Σ_u – which corresponds to the *observations* made by sensory measurements monitoring environment, or *responses* which are observations about the effects of actions performed by the subject of control;
- A set of controllable events Σ_c – which represents *commands controlling actuators* of the system, or *communicating commands* corresponding to sending and receiving messages between different components of the system,

where $\Sigma = \Sigma_u \cup \Sigma_c$.

4.1. Navigation and obstacle avoidance

Our main objective is to model cooperative multiagent behaviors, beginning with two mobile manipulatory agents each equipped with visual and other noncontact sensors. In the following section we outline models of two elementary behaviors of a single agent and their composition, and mention how this approach may be applied to modeling cooperation between two agents.

The navigation problem encompasses the control of the mobile robot moving from one location to another in a given environment. In the case where the map of the environment is available there exist a number of techniques for planning a path between two locations [11]. When a path has been determined in terms of displacements and headings (x_i, y_i, θ_i) the task of following a path

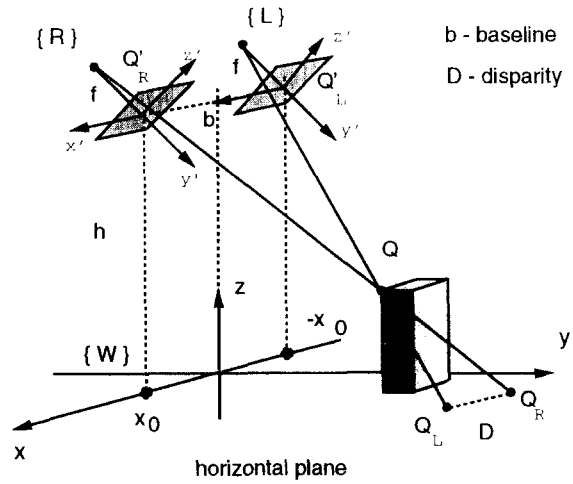


Fig. 3. Inverse perspective transformation.

in the absence of obstacles corresponds to a simple feedforward control strategy using odometry readings. This approach assumes, however, that the odometry readings reflect the correct position of the mobile agent relative to its starting position and that the path is free of obstacles. Neither of these assumptions however are reasonable in real world situations. One way to eliminate the first assumption is to use additional information (e.g. landmarks) for correcting the current heading and position of the vehicle [6]. In order to account for the presence of unexpected obstacles we need some means to detect them and realize an appropriate avoidance maneuver. For obstacle detection we have adopted an idea proposed by Mallot et al. [12]. Obstacles are detected through the difference between a pair of stereo images after applying proper inverse perspective mapping (see Fig. 3). Differences in perspective between left and right views are used to determine the presence of an obstacle and its approximate location.

First, both images from the stereo pair are mapped to the horizontal plane. If there is no obstacle above the floor, both images will be the same after applying the inverse perspective mapping. In the presence of obstacles the perspective distortions from the left and right views will be different. Therefore, by simple subtraction of the inverse perspective mappings of the left and right

Fig. 4. (a) Left image; (b) Map of the free space in lower resolution, dark areas correspond to.the free space; (c) Right image.

image one can determine the presence of an obstacle. This assumes that the movement of the mobile robot is constrained to the horizontal plane. Objects elevated above the horizontal plane are classified as obstacles. Among the numerous advantages of using this method, which is only a simple point-to-point transformation, is its real-time implementability on standard architectures. Moreover, there are no problems with shadows or textured floors.

Analysis of the free space

In our scenario the field of view of the vehicle is sufficiently large (compared to the size of the vehicle) for planning some immediate obstacle avoidance strategies.

We chose to accomplish the avoidance maneuver in a purely reactive way. However, even in this case we distinguish three stages which differ in the qualitative information extracted from the map of the free space.

1. Based on the distance and the extent of the closest obstacle in the vehicle's path, we compute the appropriate turning rate, which is proportional to the ratio of distance from the obstacle to the extent of the obstacle and to the linear velocity of the vehicle [15].
2. While the vehicle is steering away from the obstacle we monitor the position of the obstacle in the field of view.
3. When the field of view is free, the vehicle will start correcting for the accumulated deviation and eventually reach the desired path.

In the case when the information about the free space on the sides of the obstacle is not sufficient to compute the appropriate turning rate (e.g. the size of the obstacle is bigger than the field of view) we stop.

Obstacle avoidance behavior

For the scenarios proposed we assume that either the vehicle is driven by some other process (e.g. target following) or the vehicle is given a straight line path to follow. Obstacle avoidance behavior monitors the free space in front of the vehicle. When an obstacle is encountered an appropriate avoidance strategy is invoked. The DES model of obstacle avoidance behavior is illustrated in Fig. 5.

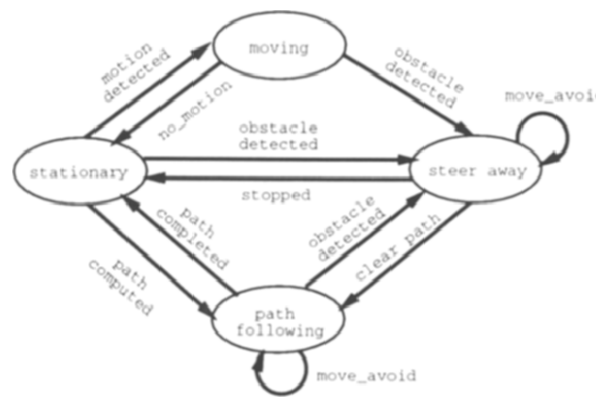


Fig. 5. Model for obstacle avoidance.

set of states $Q = \{moving, steer\_way, path\_following, stationary\}$,

set of events $\Sigma = \{obstacle\_detected, clear\_path, stopped, move\_avoid\ motion\_detected, no\_motion, path\_computed, path\_completed\}$,

where $\Sigma_u = \{obstacle\_detected, motion\_detected, no\_motion, clear\_path, path\_completed, path\_computed, stopped\}$.

initial state $q_0 = \{stationary\}$.

State 0 Vehicle is stationary.

State 1 Vehicle is moving ahead and sensors are continuously monitoring the free space in front of the vehicle.

State 2 Vehicle is turning away from the obstacle with a particular turning velocity.

State 3 Path following. The path is being followed using feedforward control strategy.

Event *obstacle_detected* is asserted when an obstacle is present in the vehicle's path.

Event *clear_path* is asserted when there is no obstacle in the vehicle's field of view.

Event *move_avoid* [2] changes the heading and/or speed of the vehicle.

Event *path_completed* is asserted when a given path is completed.

Event *stopped* vehicle stopped.

Event *motion_detected* is asserted when motion has been detected.

Event *no_motion* is asserted when vehicle becomes stationary.

The obstacle avoidance behavior is always activated when the task of the agent involves navigation. The behavior can principally run in the path following mode or exploration mode driven by a particular exploration strategy. The obstacle avoidance process is initiated in State 0. The transition to State 1 or 3 is made depending on whether the vehicle has a particular path to follow or the motion caused by the exploration (or other) process was detected. In both States 1 and 3 the free space in front of the vehicle is continuously monitored. In the presence of an unexpected obstacle (event *obstacle_detected*) the process makes a transition to State 2. After qualitative analysis of the map of the free space in State 2, an appropriate turning rate is computed and the vehicle jogs away from the obstacle. Once the field of view is free from obstacles, the process makes a transition to State 3 to pursue its original path. In case there was no original path the vehicle just makes the transition to the State 0. When the free space is insufficient to allow an avoidance maneuver, the vehicle stops (event *stopped*). From a discrete event control point of view, the obstacle avoidance behavior is controllable.

[2] *move_avoid* command encompasses the set of commands sent by the host computer to the vehicle. For details see [1].

4.2. Tracking

During the tracking mode we provide the vehicle with the capabilities of tracking a moving target. This requires choosing the right features to track and thereby uniquely describing the motion of the target in front. This problem has been addressed by several researchers [4,10], so we will focus on the issue of control.

Tracking behavior (see Fig. 6)

set of states $Q = \{exploring, tracking\}$,

set of events $\Sigma = \{target\_detected, target\_lost, move\_track\}$,

where $\Sigma_u = \{target\_detected, target\_lost\}$.

initial state $q_0 = \{exploring\}$.

State 0 Exploring. The vehicle is stationary or moving looking for the determined target.

State 1 Tracking. The target is in the field of view of the vehicle and the process generates commands to keep the target in the center of the field of view and within a constant distance from the vehicle.

Event *target_detected* is asserted when target to be tracked is detected.

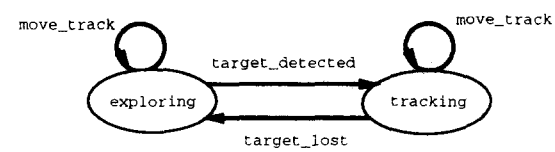


Fig. 6. Model for tracking.

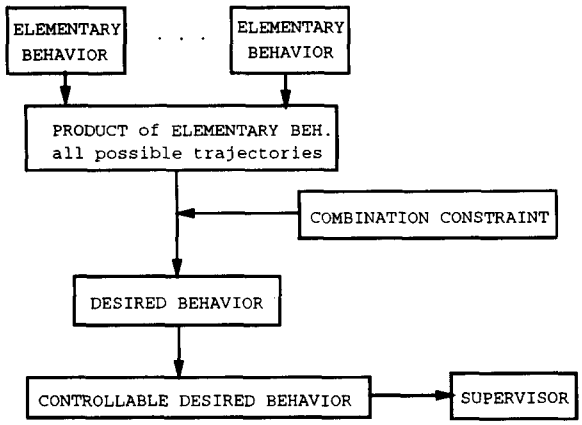


Fig. 7. Composition process.

Event *move_track* is a command to the actuator in order to keep the target in the center of the field of view.

Event *target_lost* is asserted when target disappears from the field of view.

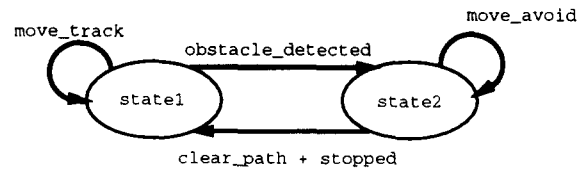
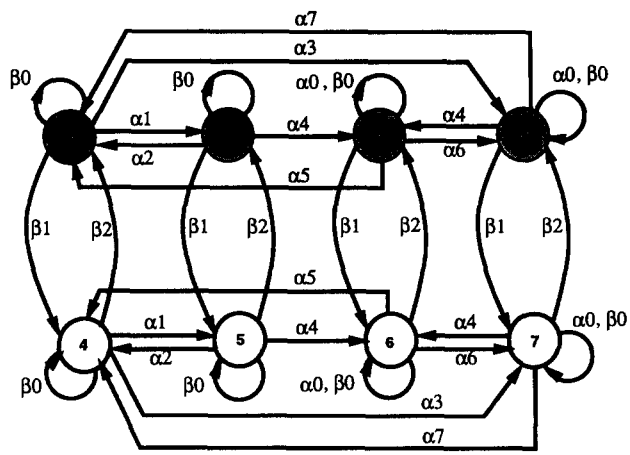


Fig. 8. Combination constraint.

4.3. Composite behaviors

The activation of different behaviors is closely related to the task to be accomplished. Composite behavior is a concurrent combination of elementary behaviors obtained using the *shuffle product* operator. Elementary behaviors are easily controllable because in each state there is only one possible controllable event which can take place; however, this might not be the case if they are invoked concurrently. Often we have to impose some constraint on composition (on concurrent operation), which can be in our case ex-



α0 move_avoid
α1 motion_detected
α2 no_motion
α3 path_computed
α4 obstacle_detected
α5 stopped
α6 clear_path
α7 path_completed

β0 move_track
β1 target_detected
β2 target_lost

$P = P_1 \| P_2$

STATE	0	1	2	3	4	5	6	7
ϕ	x x	x 1	1 0	0 1	x 1	x 1	1 0	0 1

x - don't care
1 - enabled
0 - disabled

Σ_c = {move_avoid, move_track}

Fig. 9. Product system and the feedback function Φ of the supervisor.

pressed by another automaton. The block diagram of the composition process is presented in Fig. 7. The detailed description of this process can be found in [18,14].

Obstacle avoidance while tracking

Suppose that the task is to track a moving target and at the same time to avoid obstacles. We have to activate both obstacle avoidance behavior P_1 (Fig. 5) and tracking behavior P_2 (Fig. 6). The composition of two behaviors P_1 and P_2 results in a new behavior, P, which is obtained as a product $P = P_1 \| P_2$ (Fig. 9) of the component behaviors [14]. The component behaviors are combined in an interleaving fashion, so the resulting behavior represents all possible sequences of events. The problem that may occur is that a number of controllable events (commands to actuators) are trying to control the same actuator. The role of supervisory control in such a case becomes crucial. The supervisor must now ensure that correct commands will be carried out (enabled) in response to previous observations. For example, if we consider just the open-loop behavior of the system it can easily happen that after detecting an obstacle, the agent observes that the target begins to deviate and now has the choice of correcting for deviation or avoiding the obstacle. Keeping in mind that the most important thing is to prevent the agent from crashing, we want to prevent the target following process from affecting the actuators when an obstacle is detected. This means that the supervisor should disable the event *move_track* as soon as it observes the event *obstacle_detected* until the event *clear_path* is observed. The control strategy of the supervisor for this particular example can be expressed by the automaton in Fig. 8. The on-line composition of these processes and synthesis of the supervisor is described in [3,14]. The product system and feedback function Φ of the resulting supervisor for this example is shown in Fig. 9.

4.4. Modularity vs. coupling

Up until now have we emphasized the discrete aspects of control (as formulated in Supervisory control theory) and proposed two models of simple behaviors. Particular models are completely determined by the capabilities of sensors and the types of actuators. In the above mentioned examples we dealt with a two degree of freedom system, where the only controlled actuators were the wheels of the vehicle and the supervisor's only control strategy was to switch between two processes which are trying to control the actuators. The particular hardware configuration allowed us to completely decouple the two processes and address the control issues at the level of events. In case the number of degrees of freedom of the system increases (e.g. tracking process is realized by the camera head system) the number of possible control strategies increases as well. Then the process of combining the behaviors and synthesizing a supervisor becomes more complicated. Namely, it may not be any more satisfactory to associate with the states of the supervisor only the control patterns (determining what events will be enabled or disabled). We may need to adjoin with the states of the supervisor functions which take as arguments continuous variables describing the state of the underlying behaviors (e.g. linear/angular velocity, force) and compute parameters for the commands to the actuators. It remains to be seen how the original framework can capture these system properties of interest and what are the extensions needed in order to model this class of systems (i.e. class of systems with many degrees of freedom, which are quite tightly coupled but still 'live' in the environment full of unexpected interventions and the type of coupling is task dependent). The choice of the particular semantics of communication between the plant and the supervisor stems from the physical configuration of the system as well as the type of dependencies between the modules. In the examples shown above the commands were generated by particular processes and their execution was simply enabled or disabled by the supervisor based on the previous observations. We can envision a situation when that particular observation detected in one module should trigger a transition in a different module forcing a different observation or control strategy associated with the state. This is an instance when I/O semantics applies better.

5. Conclusions and future work

Sensory guided behaviors have been of interest to the computer vision community for some time and real-time control and the role of different concurrent processes in robot control have become apparent. We believe that there is something to be gained by investigating the system as a whole rather than its separate parts. This is especially true when navigation takes place in an unknown or changing environment. While the coupling of the visual modules with navigation and observation as an active process is novel, we consider the most important contribution of this paper to be the modeling of behaviors using the theory of DES. The extraction of the right qualitative information from visual observation so that an appropriate action can take place is crucial for modeling. Moreover, the formalism we use allows us to systematically compose complex behaviors in a modular fashion and predict the controllability of the composite behaviors. The same methodology allows us to model cooperative behaviors as composite communication behaviors. It remains yet to be seen what is the right compromise between modularity vs. coupling of the sensory based behaviors, so we can preserve desirable properties of the continuous systems and at the same time allow for flexibility of composition and activation of different behaviors based on the task to be accomplished.

Acknowledgements

Navy Grant N00014-92-J-1647, AFOSR Grant 88-0296; Army/DAAL 03-89-C-0031PRI; NSF Grants CISE/CDA 88-22719, IRI 89-06770, and ASC 91 0813; and Du Pont Corporation.

References

[1] *Labmate User Manual Version 5.21L-e.*

[2] R.C. Arkin, Motor schema based navigation for a mobile robot, *Proc. Int. Conf. on Robotics and Automation* (1987).

[3] S. Balemi, G. Hoffman, P. Gyugyi, H. Wong-Toi and G.F. Franklin, Supervisory control of a rapid thermal multiprocessor (Technical report, Information Systems Laboratory, Department of Electrical Engineering, Stanford University, November 1991).

[4] B. Bhanu, P. Symosek, J. Ming, W. Burger, H. Nasr and J. Kim, Qualitative target motion detection and tracking, *DARPA Image Understanding Workshop* (1989).

[5] R.A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation* RA - 2 (1) (1986) 14–23.

[6] C. Fennema, A. Hanson, E. Riseman, J.R. Beveridge and R. Kumar, Model-directed mobile robot navigation (Technical Report COINS TR 90-42, University of Massachusetts, Computer and Information Science, June 1990).

[7] R. James Firby, Building symbolic primitives with continuous control routines, *Proc. First Int. Conf. on Artificial Intelligence Planning Systems* (1992) 62–69.

[8] M. Heymann, Concurrency and discrete event control, *Control Systems Magazine* 10 (4) (1990) 103–111.

[9] I. Horswill, A simple, cheap, and robust visual navigation system, in: *From Animals to Animats II: Proc. of the Second Int. Conf. on Simulation of Adaptive Behaviour* (MIT Press, 1993).

[10] X. Hu and N. Ahuja, Model-based motion estimation from long monocular image sequences (Technical report, University of Illinois at Urbana Champaign, 1991).

[11] J.-C. Latombe, *Robot Motion Planning* (Kluwer Academic Publishers, 1991).

[12] H.A. Mallot, H.H. Bulthoff, J.J. Little and S. Bohrer, Inverse perspective mapping simplifies optical flow computation and obstacle detection, *Biological Cybernetics* 64 (1991) 177–185.

[13] P.J. Ramadge and W.M. Wonham, The control of discrete event systems, *Proc. of the IEEE* 77 (1) (1989) 81–97.

[14] P.J. Ramadge and W.M. Wonham, Supervisory control of a class of discrete event processes, *SIAM J. Contr. Optimization* 25 (1) (1987) 206–230.

[15] C.W. Reynolds, Not bumping into things, in: *Physically Based Modelling at SIG-GRAPH'88* (1988).

[16] M.J. Schoppers, Universal plans for reactive robots in unpredictable environments, *Proc. IJCAI 1987* (1987) 1039–1046.

[17] M.A. Shayman and R. Kumar, Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models (Technical report, University of Maryland, 1992).

[18] W.M. Wonham and P.J. Ramadge, On supremable controllable sublanguage of a given language, *SIAM J. Control and Optimization* 25 (3) (1987) 637–639.

Jana Košecká received the M.S.E.E. degree in Electrical Engineering and Computer Science from the Slovak Technical University in 1988 and M.S.E.E. in Computer Science from the University of Pennsylvania in 1992. She is currently working towards the Ph.D. degree with the Department of Computer and Information Science, University of Pennsylvania, Philadelphia. Her research interests include computer vision, autonomous mobile robots and multiagent systems.

Ruzena Bajcsy received the M.S.E.E. degree in Mathematics and the Ph.D. degree in Electrical Engineering from the Slovak Technical University, in 1957 and 1967, respectively. She received a second Ph.D. degree in Computer Science from Stanford University in 1972. She joined the Computer and Information Science Department at the University of Pennsylvania as an Assistant Professor in 1972 and now serves as Professor as well as director of the GRASP laboratory (General Robotics Active Sensory Perception). She has authored numerous book chapters and journal publications and has served as editor and associate editor of several journals. Her research interest include multi-sensor integration, robotics and computer vision.