ELSEVIER

Robotics and Autonomous Systems 21 (1997) 5–21

Robotics and Autonomous Systems

Intelligent control for perceptually attentive agents: The S* proposal

John K. Tsotsos

Department of Computer Science, University of Toronto, Toronto, Ont., Canada M5S 1A4

Abstract

This proposal attempts to reconcile the successful behavior-based robot architectures with the results presented by Tsotsos (1995) which show that mechanisms which were anathema in the early behaviorist dogma, such as attention and goal-directed processing are perhaps necessary if vision is used as a major sensor. S* is a conceptual strategy for control which facilitates these mechanisms. Further, a unique method for determining whether a particular S* behavior collection can satisfy a given mission is described. A language for mission plan specification is defined permitting the integration of deliberative and reactive specifications and a proof procedure is sketched which can determine if a particular behavior set violates the timing and feasibility constraints of a specific mission.

Keywords: Control; Visual attention; Behaviors; Mission language

1. Introduction

The sensing, thinking and acting parts of a robot's function must be organized within some framework and must accommodate requirements on the robot's performance such as multiple sensors, uncertain processes, conflicting goals, and planning. Two basic control paradigms have appeared for organizing the control systems of an intelligent mobile robot: functional decomposition (or centralized) and behavior-based decomposition (or distributed). Usually the former is stated in terms of a sense–model–plan–act (SMPA) framework while the latter is given in terms of parallel sense-act layers. Hybrids of these two have also appeared.

A static, fixed SMPA model cannot react to events which occur in the robot's world while the robot is "thinking", that is, modeling and planning. This is the key reason why the strict version of SMPA is inappropriate as a control method for a robot in a dynamic world. The strict version of behavior-based control is also inappropriate because they do not lend themselves to mission-specific behavior compositions.

There are many robot control strategies; thorough reviews can be found in [1,5,9,12]. Here, we only take a look at the subsumption scheme.

The cornerstones of the subsumption architecture include control layers which define a total order on a robot's behaviors, the dominance of layers follow a hypothesized evolutionary sequence, and, each layer may "spy" on layers at lower levels and "inject" signals into them [3]. It is claimed that the structure is scalable to human-like behavior [4]. Brooks argues strongly against many of the prominent concepts and activities in AI. Specifically, he argued that there is no place for: the SMPA framework for robot control; the representation of intermediate results; the use of hierarchical computations; the explicit representation of goals; and, CAD-like models of the world. As

PII S0921-8890(97)00003-1

demonstration of his position he offers evidence that is compelling: many mobile robots that seem to have robust and interesting performance. However, a strong theoretical argument may be made demonstrating that a control strategy based on the strict version of subsumption is inappropriate for anything but a very limited domain of behavior [15]. It was shown that if scaling to human-like problems and performance is desired, behaviorist schemes must be supplemented with intermediate representations, explicit representations of goals, hierarchical organization, and attentive processes.

The proof included two theorems which specified the complexity of the stimulus-behavior (event-action) search problem. The first is: The unbounded passive stimulus-behavior search is NP-hard. It must be stressed that the NP-hardness is due entirely to the combinatorics of searching an image without an explicit target. The fact that there is additional processing for determining the applicability of behaviors and for deciding which behavior to execute only makes the time complexity of the problem worse than that of unbounded visual search [13]. Will active behaviorism help? The addition of the time dimension so that the search for a behavior-stimulus pair is over time does not necessarily help. The second theorem is: The unbounded active stimulus-behavior search problem is NP-hard. It was shown that if two problems can be solved by both active and passive methods, then the active approach will be more efficient only under certain constraints having to do with the amount of memory available for storage of intermediate results, the extent of sensor movements, the size of the visual field and so on [14]. More importantly, an intermediate representation of best hypotheses is required in order to satisfy those constraints (implied by Bajcsy [2]). The use of such intermediate representations is not within the strict behaviorist philosophy.

In addition a broader review of the control literature adds the following:

- selective attention is important for reducing computation time and can be applied to many search problems inherent in robot control decision-making [8];
- the general SMPA paradigm appears in most if not all proposals in some form [7];
- behavior decomposition gives a finer grain specification of the robot's functionality than the monolithic strict SMPA paradigm [3];
- available computation time and required response time for the robot are important variables and should be incorporated into decision-making [22];
- clustering behaviors is valuable for controlling the space of decisions required of the conflict resolution mechanism [17];
- a conflict resolution method is needed: priority rules are widely used successfully [17].

As a result, these elements form the core of the S* proposal, presented in the next section.

2. The S* proposal

In subsumption-style behavior definitions, a behavior acts directly on the physical world. In S*, this notion is generalized: the "world" on which a behavior may act may be an internal (logical) representation or an external (physical) representation. The world is added to the SMPA cycle to create an SMPA world (SMPA-W) cycle. For the remainder of this paper, a behavior is taken to mean any process which uses input available in one or more representations and causes an effect to one or more (may be the same) representations. The representations are defined as part of the agent. Each behavior is represented as such an SMPA-W cycle. Behaviors may thus act on the external world, by manipulating physical objects or causing the robot to move, or may act on the internal world of the robot. That is, a behavior may manipulate internal representations, taking input from an internal representation and making changes to or creating another internal representation for use by subsequent processes. Intermediate representations, hierarchical organizations, attentive selection, and explicit goals are thus all facilitated.

This proposal is one which subsumes the subsumption ideas of Brooks. That is, any subsumption architecture can be realized using the S* architecture, but not vice versa. This claim is not formally proved here. However, note that if the sense node simply detects a signal of a particular type, if the model and plan nodes are null, and the act node sets an actuator, the five-node cycle of S* reduces to a simple subsumption-style behavior. The priority

network required for collections of subsumption behaviors are also included in the resolve conflicts stage and with augmenting behaviors (as described later).

The working hypothesis behind this proposal follows. To date, the usual approach to achieving real-time performance has been to avoid or to minimize the role of vision. Other sensors have been used or visual analysis is trivialized in order to realize real-time operation. We are here investigating the claim that attentive, goal-driven vision also has the capacity to realize real-time performance. In trivializing visual analysis, previous researchers have implemented a primitive form of attentional selection in vision; that is, do not analyze all parts of the visual world, and only analyze to the degree required in order to achieve some goal. Although realized in an ad hoc fashion, the result is real-time performance. Here, we attempt to include attentional selection and goal-driven processing in a more principled manner and with broader scope. Further details on aspects of the S* proposal are in [16].

2.1. The SMPA-W framework

Each behavior, whether internal or external, is represented as an SMPA-W cycle. Although the concept of SMPA control has been criticized in the past, the criticism is due to its use as a single integrated control paradigm. Here, it is used as a means to decompose the stages of computation within individual behaviors. In Brook's 1986 subsumption formulation, nodes within the behaviors can be mapped onto sense, model, plan and act functions without too much difficulty particularly for the runaway and wander behaviors. The SMPA-W used here is a five-node cycle shown in Fig. 1, where the fifth node is the world which provides the input–output definition for the behavior. These behaviors include those which lead to external behavior of an agent plus those which lead to internal inferencing, reasoning or planning behaviors of the agent. The elements of the cycle are defined as follows.

World: It contains two representations, an event window and an action window (see Fig. 1). The event window opens up (or makes accessible) a relevant portion of some set of representations within the system. Similarly, the action window opens up onto some set of representations (may be overlapping with the event window). The world node also contains a set of demons which monitor the contents of the event window. These demons detect changes to the event representations of relevant types which act as triggers for the activation of the behavior. The behavior is quiet until the demons awaken it. In this way, the representations are not limited to those of the external world only and more sophisticated forms of intelligent reasoning are permitted. Intermediate representations which can be manipulated permit internal behaviors, yet are handled in the same manner as external behaviors.

Sense: This is a process which invokes a particular sampling of an event window based on active triggers in the world node, and creates a representation from that sampling. On activation of the behavior, the sense node extracts from the event window the relevant stimuli and may further process the information.

Model: Using a representation of a domain and the sensed representation, a model is associated with subsets of the sensed representation.

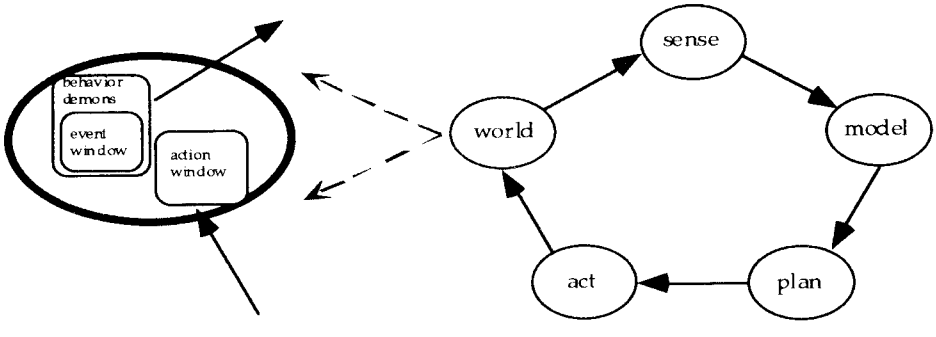

Fig. 1. The SMPA-W cycle showing the elements of the world node.

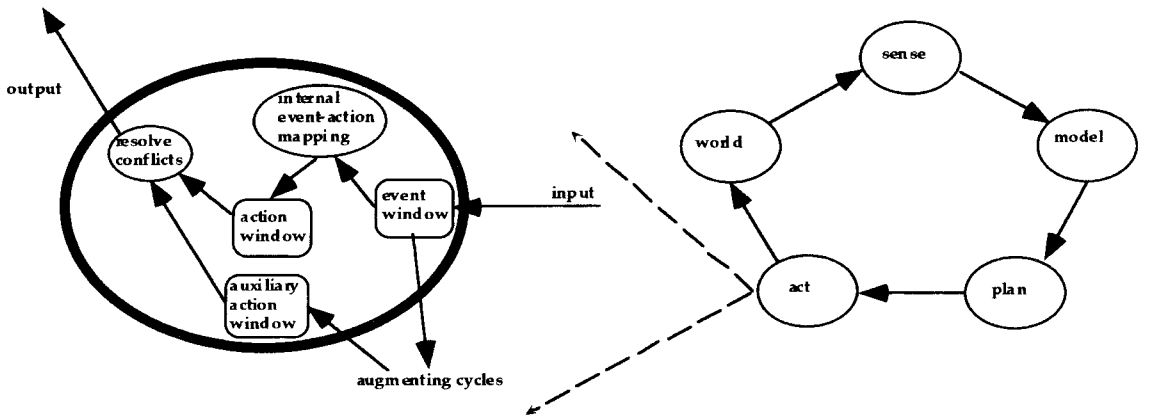

Fig. 2. The internal components of each node within the SMPA-W cycle (other than the World node which was defined in the previous figure).

Plan: This is a process which associates behaviors with models and determines the appropriateness of the association.

Act: This is a process which invokes some manipulation of a representation that realizes a behavior. All effects of the act node are seen within the action window of the world node.

This is not a strict cycle; only if the event and action windows open onto the same representation would this be so. Nevertheless, a behavior may be a continuous one; it might upon completion wait for the next triggers to appear in the event window. A single-shot behavior would terminate on completion of its act function.

The local structure within a node, shown in Fig. 2, shows each of the nodes of the cycle (other than the world node) containing the following components:

Event window: Pointers to subsets of relevant representations which contain the information necessary for the node's function. There is no restriction on the event window necessarily containing only raw signal data nor on the action window containing only actuator command representations (as in [6], for example). These representations may be any convenient ones for the behavior being implemented. In fact, the windows may open onto subsets of a single or of several representations. The event window need not be fixed but may vary dynamically. For example, although an event window may open up onto an entire camera image, it need not if an attentive process determines that some portion of the image is most likely to contain the information relevant for that behavior.

Action window: Pointers to the subsets of representations that are affected by this node's computations. This window does not change dynamically. There is no need for attention in writing, since the act of writing is specific.

Auxiliary action window: A duplicate copy of the action window which is affected by the action of an augmenting cycle which uses the same event window. There may be more than one of these; one is needed for each augmenting behavior. This is so because there is a need to compare the results of each augmenting behavior with the results of the augmented behavior in order to choose which results to pass on.

Internal event-action mapping: The processes by which the node performs the computation for which it is responsible in its SMPA-W cycle.

Resolve conflicts: If there are one or more augmenting behaviors contributing to this node then the output of the node to the cycle must be resolved. This component implements a local priority network for performing this decision. The priorities may be altered dynamically depending on current task demands or state of interpretation.

Each node in the cycle may be augmented by one or more other cycles. For example, suppose that there are different versions of optic flow computations, one which is quick but not very accurate another that is more accurate but slower. The latter would augment the former computation; a conflict resolution stage would determine which

one to use depending on task demands. Requirements for speed may lead to the former choice while a requirement for accuracy would lead to the latter. The definition of an augmenting behavior follows: an augmenting behavior is a behavior which uses the same event windows as one of the S, M, P or A nodes of some other behavior and provides competing action requests for that node. The processing done by the augmenting behavior may provide more precise or alternate descriptions using methods differing from the node which it augments.

Note that although all nodes in the control architecture have both event and action windows, it is not the case that information is duplicated all over the network. Rather, the windows may be considered as pointers into a representation stored elsewhere. In addition to economizing on storage space, there is another important reason for this arrangement. Because several windows may open onto the same representations, there must be a way of ensuring consistency within each representation. By storing only one copy of each representation, and by using windows as gateways into the representations, a single scheduler may be defined for each representation which ensures that only one node of one SMPA-W cycle can write into the representation at any one time.

2.2. *Representations for a typical S* architecture*

A number of representations would be required in any typical robot control application. A minimal set may be the following.

State. This representation contains the current state of the robot (position, orientation, camera and head information, power, global clock, etc.).

Actuator commands. This representation contains all the requests for robot motion that arise from all behaviors.

Mission/task direction. This contains the representations of the mission characteristics that come from the operator. This also contains any sub-goals determined by the behaviors to be necessary to achieve in order to satisfy the mission.

Environment. This contains the representations of the environment that the robot is in. This may include an a priori map, models of landmarks, models of known objects, etc.

Recognized aspects of the environment. This representation stores features, edges, regions, object surfaces, optic flow, model fits, recognized objects, etc., everything that is computed as intermediate as well as final results by the perception system (all sensors).

Sensed data. These are the raw signals from the robot's sensors.

Exception record (ER). Failures during the execution of a behavior must be detected; exception records encode this information. Each exception contains a specification of what must be sensed in order to confirm that the exception occurred, the identity of the behavior for which it occurred and a start time and expiry time (when to assume that no exception will occur and thus discard this exception).

Event windows (EW). The parameters of all event windows are included in the EW representation, that is, the subsets of the representations to be considered are defined.

Perception systems internal parameters (PP). The PP representation is a data base of parameters used by the perception systems and their values. This includes all thresholds, filter tuning values, any constants, etc. PP is partitioned, one partition for each sensor system. Behaviors which read PP need only consider the partition relevant to their own sensor.

How do attention, goal-direction and hierarchical processing strategies manifest themselves within S*? The three representations ER, EW, and PP play major roles. Although event windows open onto some set of representations, it is not necessarily the case that all of each of the representations need be considered at each behavior invocation. This is especially important since behaviors are activated by a set of demons which scan the event window for triggering events. Thus, any reduction of the search space is valuable. The parameters of all event windows (subsets of representations) are included in the EW representation. Event windows are dynamic and open up onto selected portions of representations. The parameters which control event windows are themselves stored in the EW representation onto which an action window may be opened. Thus, an attention behavior (or several of them)

may dynamically modify the event windows of other behaviors depending on the state of recognition or navigation progress.

A typical perception system contains many parameters, ranging from required thresholds, tuning constants of filters, to event window parameters. In an attentive selection scheme, many or all of these may be altered dynamically during the course of recognition in order to optimize the process. The PP representation is a data base of such parameters and their values. When a particular perception behavior is triggered, the current values of the parameters it requires are read in before the SMPA-W cycle is executed. The PP representation may be part of an action window for some behavior and thus the perception process itself may be tuned.

Intermediate representations appear as representations which are used in the event window of some other behavior; thus, they represent aspects of the internal world of the agent. Explicit representations of goals are included in a "mission" representation and may be used by any behavior (as part of their event window) and to target processing by any behavior. Hierarchical organization may be constructed using the flexibility inherent in the event and action window structure.

Goal-directed processing also requires a method for the detection and correction of deviations from the goal. Each "plan" node has an additional special function. Each adds a list of potential exceptions to the exception record representation. That is, these are potential failures which might occur during the execution of this behavior. Each exception must have a behavior which can deal with it, i.e., it is a trigger event for a behavior which can impose corrective actions.

2.3. *Using exceptions in an active vision recognition task*

One important task for a robot which is perceptually attentive is active object recognition, i.e., the recognition of an object using several viewpoints, where the decision to require new viewpoints is a result of incomplete or ambiguous interpretation. The new viewpoints are selected so that each assists in completing the recognition process unambiguously. This is the problem addressed in [18–20]. The overall control strategy is shown in Fig. 3 and details of each of its elements can be found in the papers cited. For example, if an object is occluded, a different viewpoint may be needed in order to fully recognize the object. This new viewpoint (may be more than one in sequence) must be determined with respect to a current state of recognition, which not only has some object hypothesis to verify, but also has a strong belief that the object is in fact occluded by something. Only then, could appropriate commands be given for motion of the camera to eliminate the occlusion.

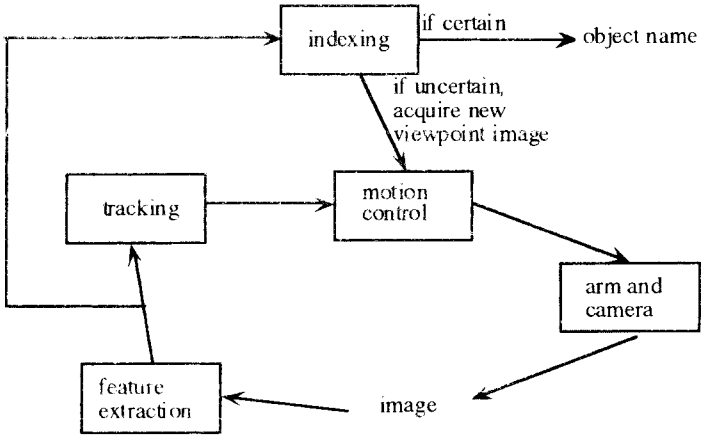

Fig. 3. Active object recognition control (from [18])

The results of the indexing process shown in Fig. 3 can be ambiguous, i.e., more than one object model has the same set of indexing features. If so, a new viewpoint is selected which would hopefully dis-ambiguate the object matching. How can such a control strategy be represented in S*? The failure of unambiguous matching at this step can be represented using an exception. The event which would trigger the exception is that more than one object hypothesis is active and none of the hypotheses have a sufficiently strong confidence level. The behavior which is triggered by this event would determine a new viewpoint and send a request to the camera system for the acquisition of a new image. The new viewpoint is selected as a function of the best few object hypotheses, the differences and similarities among those hypotheses, and special viewpoints which are encoded in the system for each object (correspondence between actual and stored poses must be computed) which are known to provide certain feature information (see [18] for details). The recognition strategy would then be re-started with the new image, but with the context that the image is one of a sequence in the recognition process. This strategy would require two interacting behaviors, active object recognition and select new viewpoint.

The active object recognition behavior has a sense node which detects that an unlabeled visual item of interest is in the scene, and then would extract visual parameters for that item. The model node would select the most likely models from the model base using the features as indices and then match the visual data to those models, noting which are the best matches. The Plan node would add exception records to the ER representation which would be activated if the features extracted are not of sufficient quality (the item may be out of focus or is too far or too close) or if there are multiple good matches. Each of these conditions would lead to selection of a new imaging geometry. The act node would make changes to the ER representation as well as to the representation which stores recognized characteristics of the environment.

Select new viewpoint is activated when one of the above exception conditions is satisfied (note that many other behaviors might also activate such exceptions requiring this behavior to select new viewpoints for those processes as well). The sense node notes which exception type and which behavior activated it and extracts the relevant information. The Model node uses the current best hypotheses plus their stored models to determine the next best viewpoint. The plan node determines the sequence of movements to bring the imaging geometry in line with that required. The act node then makes the appropriate changes to the ER and other representations. The plan node must also add potential exceptions to the ER representation; for example, an exception would be raised, requiring a new viewpoint, if the imaging geometry cannot be achieved due to internal or external constraints.

This strategy has been successfully implemented for a model base of a dozen origami objects [18] without the S* formalism.

5.4. A rough guide to building an S* control structure

It must be clear that the intent of S* is not to provide an optimal controller in the control theoretic sense. In any case, the simultaneous optimization of the large number of variables associated with a perceptually attentive robot system is quite far beyond the current state of control theory research. Rather, S* is intended as a conceptual framework which seeks to remedy the representational shortcomings of the subsumption architecture. This section presents a guide to building an S* structure for a particular robot. The guide suggests which behaviors and representations must be defined as a minimal set. This section also describes a language for defining a mission and provides a proof procedure for rejecting behavior sets for a particular mission which could not satisfy timing constraints.

There exists a vast array of formal methods for specifying, developing and verifying real-time systems (see [11] for comprehensive review). The general conclusion is that real-time systems are difficult to model, specify and verify and thus any solutions will not come easily. Formal methods are pursued because it is believed that they may be verified mathematically. Further, the process of formalization itself is valuable since it may reveal inconsistencies and omissions.

In order to prove a particular algorithm correct, one requires a language for its representation, a specification language for expressing properties (the list of requirements that system must ideally satisfy), and a proof system for verification. Further, one should provide a formal semantics so that the behavior of the program and meaning

of specifications are clearly defined, the proof system must be sound with respect to those semantics so that only algorithms whose behavior satisfies the specifications will be proven correct, and the proof system must be complete, i.e., all algorithms which satisfy the specifications are provably correct. Usually, a real-time system can be decomposed into a controller and a plant; both must be verified for correctness. In general, this is impossible. Any formal theory must address the modeling (description of the system complete with all dynamic and temporal properties), verification (proving that the system satisfies a given task specification), and development (tools to assist in the construction of the controller so that the system may satisfy a given specification).

The remainder of this section makes several steps towards providing these capabilities for S* but does not yet fully satisfy the above requirements.

The starting point for creating a generic S* structure is the definition of all the representations, behaviors, exceptions and trigger events for each behavior. As stated earlier, the representations for exceptions (ER), perception parameters (PP) and event windows (EW) are mandatory in S*. It is assumed that controllers exist for variables such as speed, acceleration, etc., for each actuator sub-system. A high level procedure for defining the behavior set is:

(1) Enumerate the actuators of the robot.
   - For each, define a behavior to arbitrate the requests to it.
   - Define a representation for those requests.
(2) Enumerate the classes of actuator actions.
   - For each actuator action class, at least one behavior places those action requests.
(3) Enumerate the sensors.
   - At least three representations are needed for each sensor: one to hold the raw sensed data, one to hold the interpretation of that data and one for the perception parameters.
   - For each sensor, at least one behavior must provide an interpretation of the data.
   - For each sensor, at least one behavior which sets its perception parameters.
(4) For each behavior, enumerate all exceptions.
   - Each exception requires at least one behavior which can deal with it.
(5) For each behavior, determine trigger events, include exceptions which are triggers.
   - For each trigger, at least one behavior writes the trigger data into some representation.
   - For each trigger, define an appropriate world node demon for its detection.
(6) For each representation, there must exist at least one behavior which reads it.
(7) For each behavior define appropriate sense, model, plan, act and world nodes.

Behaviors and representations are defined independent of one another, with the intent that they operate in parallel and autonomously. The link among behaviors is the set of representations on which they act.

3. A mission specification language

Execution of a mission requires a first stage to define the task steps. The intelligent agent task is more difficult than single-mission systems since the agent must be capable of executing a wide variety of mission types. If the mission specifies that the robot must move from one point to another on a map, then a simple plan is possible in terms of the motions that the robot must execute (say in terms of move forward, left, turn, etc.). Also, the planner would know the kinds of position uncertainties inherent in the robot's movement, and could explicitly include position verification and correction steps into the plan. Reactivity is not so easily captured by the STRIPS-like list of plan steps implied here. What is really required is a new method for specifying a plan that includes not only the motions of the robots, but also specifies the aspects of the environment of which the robot must be vigilant while executing the plan. These include things like "ensure that your position is within a given tolerance", "do not hit anything", "give the right of way to humans", etc.

In order to accomplish this, suppose that a simple modification is made to a STRIPS-like plan language. Add a grouping construct and permit nesting using this construct, so that for a collection of plan steps, actions may

be associated with that collection to ensure that a particular Boolean expression is true. Call this construct the "while...ensure" primitive. The computation involved in checking a constraint takes time and it may be that some constraints must be continuously verified while others perhaps only infrequently; the ensure clause requires a specification of "rate", that is, how often must the elements of the Boolean expression be verified? As a result, both deliberative and reactive aspects of the plan are fully laid out. Further, the Boolean expressions may offer a way of changing the optimization parameters of the system within a mission. Consider the following:

```
while { /* Phase 1 */
        move forward 10 feet
        turn left
        while { /* Phase 2 */
                move forward 25 feet
                find landmarks A, B and C
                verify position
                move forward 30 feet
                ensure (2/min, ¬yield-right-of-way && position-uncertainty < 1m
                        && avoid(fuel spills))
        }
        /* Phase 3 */
        turn right
        move forward 5 feet
        ensure (5/min, yield-right-of-way && position-uncertainty < 1.5m)
}
```

There are three major phases to this mission. The first and third are governed by the ensure statement at the end of the first "while", while the second has its own ensure statement. Note how the inner ensure Boolean over-rides some of the predicates of the outer Boolean and adds an additional condition (avoiding fuel spills). During the second phase of this mission, the region to be traversed by the robot might not have humans or it may be far more important for the robot to avoid the spills and thus it is instructed not to yield right of way. The tighter constraint on position also implies that any unnecessary motion should be avoided.

The ensure clause has a second functionality, namely, that of enabling automatic mission execution monitoring. A user needs to include the aspects of verification for mission execution that are necessary.

It is now possible to specify a grammar that gives the syntax of a well-formed mission. The examples are re-defined by this grammar:

mission ← **start** behavior-list **end**
behavior-list ← behavior
| behavior-list behavior
| **while** { behavior-list **ensure** (rate, Boolean) }
rate ← rate-specification

Keywords of the language are in bold font. Each phase is a mission goal; each of the steps of a phase are its sub-goals. Each is sequenced by the temporal connectives which are presented next.

There may be a need for more sophistication in mission specification than is available in this simple, single behavior line method. After all, S* permits all behaviors to be active in parallel (at least the event demons of the sense nodes of all behaviors are active at all times). Thus, for complex missions, there is a need to specify parallel phases of the mission that go beyond what is specifiable using the ensure clauses. Define a new construct called a "phase" which is used to form a group of mission steps ("atomic" steps such as those described so far, while...ensure groups, as well as other phases).

A temporal specification language would be useful here since with parallel behavior execution, the timing of behaviors is important (think about the need to confirm that a particular trigger event will actually be available when it is needed in order to motivate this need). Several have appeared previously. What is needed is a simple structure to connect events in time. If phases are considered as mission steps, then we can specify the temporal connections and constraints among phases. Note that the following of a temporal link is sufficient to awaken the demons to their vigilant state in a behavior's world event window.

One further extension is needed. The above permits pairs of behaviors to be related in time but there is no mechanism specifying, for example, that a group of behaviors must all begin at the same time or that a group of behaviors has some particular temporal relationship with another group. In order to permit this, a second grouping construct is needed, the list. A list of behaviors may be aligned in time by their starting time or by their expected end time. The behaviors are delimited in the list using commas. Behaviors aligned by start time are represented as [b1, b2, b3) and by end time (b1, b2, b3]. Finally, an upper bounds time constraint may be associated with a mission and/or with each phase. A semi-colon as behavior connector implies the two linked behaviors execute one after the other. The expanded grammar becomes:

mission ← **start** time behaviors **end**
behavior-atom ← behavior-id (p-list)
| phase-id
| while
behaviors ← behavior-atom
| behavior-list
| behaviors behavior-connector behaviors
phase ← **phase** phase-id time { behaviors }
while ← **while** while-id { behaviors **ensure** (rate, Boolean) }
rate ← null
| rate-specification
time ← null
| time-specification
behavior-connector ← ;
| temp-connector
behavior-list ← [b-list)
| (b-list]
b-list ← behavior-atom
| b-list, behavior-atom
temp-connector ← **starts with**
| **within**
| **during**
| **starts after** time
| **ends with**
| **ends during**
behavior-id ← { system behaviors }
p-list ← parameter
| p-list, parameter
parameter ← { possible system parameters }

A hypothetical example follows which builds upon the previous one.

start < 75 min
phase P1 { find landmark (D);

```
                move forward ( 10 feet ) within track landmark ( D );
                turn left
}
phase P2  {     move forward ( 25 feet );
                find landmarks ( A, B and C ) starts with verify position;
                move forward ( 30 feet )
}
phase P3  {     turn right;
                move forward ( 5 feet )
}
phase P4  {     find landmark ( E )
}
while W1 { P1 starts after P4
        while   W2 { P2
                    ensure (2/min, ¬yield-right-of-way && position-uncertainty < 1m
                            && avoid(fuel spills))
        }
        P3 < 30 min
        ensure (5/min, yield-right-of-way && position-uncertainty < 1.5m)
}
end
```

Behaviors have visual targets; in phase 2, the second behavior is to find landmarks A, B, and C. Other targets would be fuel spills, people and other vehicles for the ensure clause of phase 2. It is assumed that procedures for the recognition of targets exist and are coded as behaviors. There are also implicit visual targets; these are not directly specified by the mission, but which are part of the proper execution of a behavior specified by the mission. Each of the "move forward" behaviors, for example, must ensure that there is a clear path available before the motion is executed.

The sequence of behaviors forms a directed graph and an important computation must be performed on this graph (described below). However, details on how to construct this graph are not presented here (see [16]). Note that the closed world assumption is not made; this makes the derivation of proofs regarding the completeness and formal semantics of this temporal specification impossible. What follows in the next section is how specifications of missions are transformed into sets of behaviors. Although this simple example may appear complete, it is not necessarily so and further work on this type of mission specification is needed.

4. From mission specifications to behavior sets

How can a set of behaviors be selected to satisfy a particular mission specification? Suppose a mission must be completed in time τ. This might be a more difficult problem if it were not for a particular simplifying tactic: the mission specification language uses the behaviors themselves as part of the language; this yields the initial set of members to the behavior set B. Although this solves part of the problem, it does not solve all of it. The behaviors specified in the mission might require many additional supporting behaviors in order to properly execute. A simple method may suffice for determining which those supporting behaviors are:

Repeat until no new behaviors are added to set B.
For each behavior, enumerate "world" node triggers (including representations where they are to be found).
For each, locate the behaviors which provide those triggers to those representations.
Add those identified behaviors to B.

All the information required for this procedure is available as a result of the procedure outlined in Section 2.1 for defining the set of behaviors for a particular robot. Behaviors that provide trigger events must be ordered in time; this ordering would be at best a partial order. The ordering must have links into the mission sequence so it is known when a behavior's trigger events might be available. Thus, a network can be created whose nodes are behaviors, each node is labeled with its worst-case cost in terms of computation time as a function of input size, and directed links between nodes show the sequence of behaviors specified by the mission as well as the temporal relationships among those behaviors and the behaviors added in order to provide trigger information.

Behaviors may be of two types: single-shot and continuous. Most of the above applies for a single-shot behavior. A continuous behavior, may be either of the kind implied by the SMPA-W cycle or the "while-ensure" clause in the mission specification. Any behaviors invoked by the "ensure" part of the latter clause would have a cycle time that is very short in comparison to the behaviors to which they are attached and it is useful to specify a sampling rate for the condition in order to be certain that the ensure clause is executed appropriately (the "rate" parameter shown above). All other continuous behaviors cycle through at a rate determined by their execution time (t_b defined in the next section). It will be assumed that new results due to the behavior appear in the output representations with this rate. The remainder of the discussion will focus on methods for verification of mission timings based on sequences of single-shot behaviors; continuous behaviors are included via the ensure clause only.

4.1. Confirming behavior sets for particular missions

It is important to show that a given behavior network, derived from a mission specification, can actually satisfy the timing constraints imposed by the mission. In general this is very difficult especially if the range of unexpected situations which real world environments present is considered. In order to attempt a solution to this problem, the possibilities are divided into four scenarios, from simplest to most complex. Only for the simplest is a satisfying solution available.

Scenario 1. The proof-theoretic procedure below accomplishes this on the assumption that no exceptions are raised during the execution of any behavior set. Clearly, if exceptions arise, no timings can be predicted. Further, it is assumed that the ensure clause corrective behaviors are never activated and that all trigger information is available when needed. Most of the machinery presented below will also be relevant for the remaining scenarios.

Each behavior b has associated with it the following:

$C_b(e_b)$	cost (in time) of computation as a function of size of input e_b (event window);
W_b	cost (in time) of writing to action window;
A_b	incremental additional cost (in time) of augmenting behaviors (plus conflict resolution);
$e_{b_{\min}}$	minimum event window size required (defined by recognition targets, imaging parameters).

The costs are all worst-case (or upper bounds) estimates which must be derived by analysis of the algorithm and code included in each behavior. This does not take into account any time used waiting for a trigger event or for other data to be provided by some supporting behavior nor should it; the network structure includes this implicitly. The amount of time which may be wasted waiting for read and write permissions is not included. It will be assumed to be zero.

The reason why this particular formulation was chosen has its roots in the reason for wanting to enhance subsumption in the first place. The arguments laid out in [15] were based on computational complexity and demonstrated that the subsumption architecture was inadequate because it ignored complexity issues particularly as they pertain to sensory processing. It seems natural then to include complexity directly in S* in this manner in order to overcome this inadequacy.

The specification of $e_{b_{\min}}$ is motivated by the study of 3D search behavior in [21]. There a detailed algorithm is presented which outlines how visual search might take place in an unknown and large 3D space. One of the important sets of parameters required by the algorithm is the field of view in relation to the target sought. Any

recognition algorithm must be presented with a reasonable projection of potential targets in images in order to have some chance at detecting them. A simple illustration will suffice to motivate this. Suppose that a landmark, say a red triangle, is to be found in some 3D space. In this space there are many other polygonal shapes as well as red and other color objects. Although the attention of the system may be drawn to red items as a first pass determination of candidate locations to inspect, the red triangle might be so far away from the robot that its image subtends only a few pixels in the image. The only recourse for inspection is to either zoom in on the candidate, thus enlarging the image, or to move closer with the same effect. Thus, $e_{b_{\min}}$ is set for each trigger event using the algorithms and methods outlined in [21].

The total time to run a given behavior is

$$t_b = C_b(e_b) + W_b + A_b.$$

Most behaviors seem to have a function C_b which is either constant or has very small variation with input size. This is particularly true in situations where the input is some small list of information. The function achieves its full importance if sensor data are concerned and where search over large spaces is required. e_b is the actual event window size required and is dependent on the actual targets specified for the behavior.

If a specified task T must be accomplished within a particular time interval τ, it is possible to determine for a given set of behaviors what event window sizes might be required. It is even possible that a particular behavior set can be rejected as being infeasible for the task. In its simplest form, if the event window allowed by the timing constraint is smaller than the minimum required event window, the behavior is infeasible. In more complex forms, there is a hierarchy of timing constraints that needs to be satisfied. The hierarchy is determined by the nesting of the behaviors via the "phase" and "while...enusre" constructs. Each phase or ensure condition might have its own timing constraint. Thus, not only must the overall constraint τ be satisfied, the constraints on each subset of behaviors also must be satisfied. Let the set Δ be the set of such behavior subsets, the ith element is denoted by δ_i each with their own timing constraint τ_{δ_i}. Then the following must be satisfied:

$$\forall i, \quad \delta_i \in \Delta, \quad \tau_{\delta_i} \leq \sum_{b \in \delta_i} t_b.$$

Strictly speacking, this assumes that the behaviors in each subset are single-shot behaviors in sequence. This is not in general true; the behaviors may form a network and in this case, the longest path is what is actually needed above. Even so, the above does provide a conservative estimate on this constraint.

Suppose that a set of behaviors B has been selected to satisfy T, that network is created whose nodes are these behaviors, and each node is labeled with cost using $e_{b_{\min}}$. Let the longest path within this network of behaviors be Φ. Although the general problem of finding the longest path in a graph in NP-complete, polynomial time algorithms exist for acyclic directed graphs [10]; Φ is such a graph. It is possible to compute an overall minimum (but not actual) duration D for the execution of the task in terms of the sizes of inputs to each behavior:

$$D = O(B) + \sum_{b \in \Phi} [C_b(e_{b_{\min}}) + W_b + A_b],$$

$O(B)$ is the overhead required for execution of the overall system as a function of the behavior set (event demons, database accesses, real-time operating system, etc.). Although an estimate for the duration of the set Φ may be found by using the times to execute determined by $e_{b_{\min}}$ for each behavior, when a particular e_b is assigned (as a result of target or trigger event information), the duration will change.

A third complexity constraint can also be stated based on the network linking the behaviors of set B. Any real system will be constrained not only by time but also by space. In particular, the number of behaviors which must be active in the system at any time must be such that the available processors and memory are not exceeded. For this discussion, it is assumed that each behavior requires a dedicated processor and that the processor must have a specified amount of memory for the execution of the behavior. Assume that the set of processors available is

P. At any point in the behavior network, it is possible to identify which behaviors are expected to be active; each behavior has an expected execution time t_b which will be a function of the behavior's $e_{B_{\min}}$. Further, the memory requirement of each behavior will also depend on $e_{b_{\min}}$ for that behavior. Partition the network of behaviors into sets such that each set contains either a single behavior that executes or a set of behaviors which execute in parallel. Let the set of these sets be Σ. The number of processors needed would be simply the cardinality of the behavior set s, where s is an element of Σ. Thus, the maximum number of processors required for the mission may be given as $P = \max_{s \in \Sigma}(|s|)$ while the memory requirements are: $M = \max_{b \in B}(R(b, e_b))$ where R is the amount of memory behavior b requires on the processor where it executes when its event window is of size e_b and $P_{\max}$ and $M_{\max}$ are the absolute maxima for these quantities.

The next step is to determine the best values of e_b using the specific targets and trigger events given in the mission. If, in order to satisfy the time constraint, the size of any behavior's event window must be smaller than the corresponding $e_{b_{\min}}$ for that behavior, then the set B is not a feasible solution for T. The search problem is then re-cast as the following:

(1) choose the elements of set B and determine the sets Φ (based on $e_{b_{\min}}$), Σ and Δ
(2) $\forall b \in B$, find $e_{b_{\min}}$,
   (a) such that $e_b \geq e_b$,
   (b) $\forall i, \delta_i \in \Delta$, $\tau_{\delta_i} \leq \sum_{b \in \delta_i} t_b$, (defined using e_b),
   (c) for D defined using e_b, $D \leq \tau$ and is as small as possible,
   (d) $P \leq P_{\max}$,
   (e) $M \leq M_{\max}$.

A solution may be obtained for this problem via dynamic programming; the full optimization problem is likely NP-hard. Note that this procedure can only be used to conclude that a particular behavior set B *cannot* satisfy the timing constraints. If the set B passes this test, the only conclusion that can be drawn is that the resulting system *may* satisfy the timing constraints. The fact that exceptions cannot be included in this procedure is the main reason for this. It is important to note that this does not check the correctness of the behavior set for the specific mission.

Scenario 2. In this scenario, it is not assumed that triggers are available when needed. All the above machinery applies if it can be assumed that a model of the expected wait time can be empirically determined. The expected wait time is then summed with C_b.

Scenario 3. The execution of the corrective behaviors due to the ensure clauses is now added. These behaviors are continuous ones; if triggered by failures of constraints they execute and then return to a vigilant state. Therefore, strictly speaking loops have been introduced into the behavior network. If it is possible to empirically determine expected delays due to corrective behaviors, then the same simplification can be made as Scenario 2.

Scenario 4. Finally, exceptions are included. Although some upper bound may be added in as in Scenarios 2 and 3, it is, in general, unreasonable to expect that any timing guarantees can be provided if real world problems are permitted. An uncooperative element (human or otherwise) can delay a mission indefinitely.

Is it possible then, if a given set B does not satisfy its timing constraints to find an alternate set B' which does? This is unlikely for this particular formulation. This is due to the fact that the mission, specified by the system operator, itself defines the main sequence of behaviors which must be included in B. The supporting behaviors for B are the only ones which S* adds. There might be some room for alternative recommendations if very time consuming augmenting behaviors are part of the plan. It may be that their removal would allow the modified set B' to satisfy the timings needed. On the other hand, the removal of those expensive behaviors would presumably cause other dimensions of the mission to be unsatisfied.

This coarse guide to behavior set selection can be refined in several ways, not all explored in this document. For example, if the mission specification includes information regarding the quality of, say, recognition results, and it each behavior includes a specification of the quality attainable with the behavior and with each of its augmenting behaviors, then the comparison of these quantities may lead to a decision on whether or not to include the augmenting behavior in the final set.

4.2. Comparisons to other formal methods

One of the contributions of this work is the use of complexity-theoretic time measures in the specification of a behavior and in the verification of behavior sets for a particular mission. Previous schemes use a variety of mechanisms for dealing with time. Some employ the "synchrony hypothesis": this assumes that the system instantly reacts to external events and does not take into account the finite time required for computation. "Statemate" is an example of a popular tool which makes this assumption. Another tools is the use of upper and lower bounds on time specifications, such as are used in timed Petri nets. Here, events may occur anywhere within an interval and still be correct. Logical formulations (using formal logic) also typically employ time bounds, and many also use external clocks to which the bounds are tied.

Formal methods usually must assume that the external world is known, closed and totally predictable. This is not the case in real situations. Otherwise, the proof procedures cannot be applied.

It is interesting to note that in all case, these formal methods lead to verification procedures which are provably exponential (oven doubly exponential – see [11] for a useful comparison). In order to make matters worse, none of the formal methods makes the proper assumptions regarding perception. That is, perception is assumed to be at worst a linear problem, when in fact, it might be quite a bit more expensive if not also exponential in nature.

In the above method for S* a new timing tool is proposed which acknowledges that behaviors have different time requirements for different tasks. In other words, rather than using fixed lower and upper bounds irrespective of the task, S* permits a more fine-grained analysis of timing requirements by using a complexity-theoretic measure, which is specific for the amount of input required for a particular task.

Although a complexity-theoretic timing bound has important advantages, it does not change the fundamental exponential nature of the proof process. Here, the optimization problem is cast into a format where dynamic programming may find approximately optimal solutions. The procedure simply checks a particular behavior network derived from the mission specification; if it satisfies the timing constraints, then it may be termed "correct" (at least, this is what all other formal methods conclude; in reality, it is only correct as far as timing is concerned). Search for the optimal behavior set is outside this procedure.

The method is novel, provides for fine-grained task-specific analysis of behavior timing and makes no trivializing assumptions about the computational costs of any of the solution's components. Further research is required in order to experimentally verify that the method has practical utility.

5. Conclusions

A new conceptual structure for intelligent, visually guided, robotic control has been presented. The strategy was motivated by the current successful methods (such as subsumption) but also by a theoretical analysis of those methods which showed that many of the computational mechanisms previously thought unnecessary are actually quite important (if not critical). These include visual attention, intermediate (shares, internal) representations, hierarchical processes, and goal-driven processing. The S* architecture facilitates these mechanisms within a behavior-based framework.

To date, the usual approach to achieving real-time performance has been to avoid or to minimize the role of vision at all cost. Other sensors have been used or visual analysis is trivialized in order to realize real-time operation. We are here investigating the claim that attentive, goal-driven vision also has the capacity to realize real-time vision. In trivializing visual analysis, previous researchers have implemented a primitive form of attentional selection in vision; that is, do not analyze all parts of the visual world, and only analyze to the degree required in order to achieve some goal. Although realized in an ad hoc fashion, the result is real-time performance. S* attempts to include attentional selection and goal-driven processing in a more principled manner and with broader scope.

The behaviors of S* are generalized in the sense that they may operate either on the physical world or on internal representations and these types of behaviors are treated uniformly. An important component to the overall structure

is a set of representations that includes definitions of event and action windows for each behavior (where to look for events that activate a behavior and which representations are affected by the behavior), representations for perception parameters (event and action window parameters as well as filter, recognition tunings, thresholds, etc.) and exception records (workspace which records exceptions that might occur during behavior execution so that failures may be identified and remedial action taken). Any behavior may manipulate these representations in order to tune the overall system to particular short or long term goals.

In addition to presenting additional conceptual power in the form of the S* architecture, a unique method for determining whether a particular S* behavior collection can satisfy a given mission is described. A language for mission plan specification is defined which permits the integration of deliberative and reactive mission specifications such that all parameters specifying both types of robot behaviors may be dynamically under the control of the mission. Finally, a proof procedure is sketched which will confirm or deny that a particular behavior set satisfies the timing and feasibility constraints of a specific mission plan. To our knowledge, this is the first such proof procedure designed that ties mission plans to particular control architecture instantiations and mission timings.

All the specifications of S* are preliminary; that is, there have been no experimental verifications of any part described in this document. The overall design of S* is novel and promises much greater flexibility, scalability, and more intelligent handling of the perception component for intelligent robots. This latter aspect seems most lacking in other current work on this topic.

Acknowledgements

The author thanks Allan Jepson, Michael Jenkin, Evangelos Milios, David Wilkes and Piotr Jasiobedzki. This research was funded by PRECARN Associates Ltd., Technology Ontario, Ontario Hydro and AECL Ltd.

References

[1] R. Arkin and R. Grupen, Behavior-based reactive robotic systems, Tutorial Notes, *IEEE Conf. on Robotics and Automation*, Atlanta (1993).
[2] R. Bajcsy, Active perception vs. passive perception, *Proc. IEEE Workshop on Computer Vision: Representation and Control*, Bellaire, Michgan (1985) 55–62.
[3] R. Brooks, A layered intelligent control system for a mobile robot, *IEEE Journal of Robotics and Automation* RA-2 (1986) 14–23.
[4] R. Brooks, Intelligence without representation; *Artificial Intelligence* 47 (1991) 139–159.
[5] R. Chatila and S. Harmon, eds., *IEEE Workshop on Architectures for Intelligent Control Systems*, Nice (1992).
[6] J. Connell, A Colony architecture for an artificial creature, MIT AI LAB, Ph.D. Thesis, AI-TPP51, 1989.
[7] C. Fennema, A. Hanson, E. Riseman, J. Beveridge and R. Kumar, Model-directed mobile robot navigation, *IEEE Transcations Systems, Man, and Cybernetics* 20 (6) (1990) 1352–1368.
[8] P. Grandjean and L. Matthies, Perception control for obstacle detection by a cross- country rover, *Proc. IEEE Int. Conf. on Robotics and Automation*, Vol. 2 (1993) 20–27.
[9] S. Iyengar and A. Elfes, *Autonomous Mobile Robots: Control, Planning and Architecture* (IEEE Tutorial Press, New York, 1991).
[10] E.L. Lawler, *Combinatorial Optimization: Networks and Matroids* (Holt, Rinehart and Winston, New York, 1976).
[11] J. Ostroff, Formal methods for the specification and design of real-time safety critical systems, *Journal of Systems and Software* 18 (1) (1992) 33–60.
[12] C. Thorpe, Robots, Mobile, in: S. Shapiro, ed., *Encyclopedia of Artificial Intelligence*, 2nd Ed. (Wiley, New York, 1992) 1409–1416.
[13] J.K. Tsotsos, The complexity of perceptual search tasks', *Proc. Int. Joint Conf. on Artificial Intelligence*, Detroit (1989) 1571–1577.
[14] J.K. Tsotsos, On the relative complexity of active vs. passive visual search, *International Journal of Computer Vision* 7 (2) (1992) 127–141.
[15] J.K. Tsotsos, On behaviorist intelligence and the scaling problem, *Artificial Intelligence* 75 (1995) 135–160.
[16] J.K. Tsotsos, Intelligent control for perceptually attentive agents: The S* proposal, RBCV-TR-96-46 (1996).
[17] M. Watanabe, K. Onoguchi, I. Kweon and Y. Kuno, Architecture of behavior-based mobile robot in dynamic environment, *IEEE Robotics and Automation*, Nice (1992) 2711–2718.
[18] D. Wilkes, Active object recognition, Ph.D. Thesis, Department of Computer Science, University of Toronto (1994).

[19] D. Wilkes, S. Dickinson and J.K. Tsotsos, Quantitative modelling of view degeneracy, *Proc. 8th Scandinavian Conf. on Image Analysis*, Tromso, Norway (1993).
[20] D. Wilkes and J.K. Tsotsos, *Active Object Recognition*, CVPR-92, Urbana, II (1992) 136–141.
[21] Y. Ye and J.K. Tsotsos, Sensor planning in 3D object search, RBCV-TR-94-47 (1994).
[22] S. Zilberstein and S. Russell, Anytime sensing, planning and action: A practical model for robot control, *Proc. IJCAI*, Chambery, France (1993) 1402–1407.

John K. Tsotsos was born in Windsor, Ontario. He received his Ph.D. in 1980 from the University of Toronto in Computer Science. He currently is Professor in that department and maintains a status Professorship in the university's Department of Medicine. He has served on numerous conference committees and the editorial boards of *Computer Vision and Image Understanding, Computational Intelligence* and *AI & Medicine*. His research focusses on biologically plausible models of visual attention, the development of a visually-guided robot to assist physically disabled children and perceptually-guided robot control mechanisms.