An Architecture for Action with Selective Attention

John Woodfill
Ramin Zabih

Computer Science Department
Stanford University
Stanford, California 94305

Abstract

We believe that an autonomous system which acts in the world will need to concentrate its attention on a subset of the objects in view. The mechanism for selective visual attention will be a critical part of such a robot — it can hardly act correctly if it focusses on the wrong objects. In order to explore these issues, we have designed a system which utilizes selective visual attention, based upon both psychological evidence and computational constraints. Our system performs well in an interactive domain with several hundred objects despite tracking only a few of them at a time. We describe our implementation of attention and attempt to account for our system's performance.

Area: Robotics
Subtopic: Architectures for situated activity

1 Introduction

How could a robot drive a car? More broadly, how is it possible for a system to deal with a complex, unpredictable domain? Planning is the traditional method that AI has proposed: given a theory of the domain, a system is expected to use some form of automated deduction to decide what to do next. However, this notion of planning suffers from two major problems: it is computationally intractable, and hence is difficult to apply to domains that require real-time response; and it requires an explicit, correct axiomatization of the problem, which is unlikely to be available for such domains. Recently, several researchers such as Brooks [4] have proposed an alternative. Their idea is to build real-time systems that produce a reasonable response in every situation. We are largely in sympathy with this approach.

To build a robot with a chance of surviving in a complex domain, one would have to address the issue of focus. If a car-driving robot were to spend an equal amount of time worrying about each thing it could see, it would rapidly find itself in trouble. It is hardly novel to claim that focus is important, but we believe that selective attention, and particularly selective *visual* attention, will be critical for autonomous systems. A system without selective visual attention would need at the very least to recognize all the objects in every scene. Its performance would necessarily degrade as the number of visible objects increased, which we believe would be impractical. We wish to explore the issue of selective visual attention, and its relationship with action.

We begin with a general discussion of visual attention, and present a model of attention motivated by both psychological and computational constraints. Our model proposes that a small fixed number of objects is recognized and tracked, and that certain "pop-out" events outside of these objects can attract attention. In section 3 we describe a system we have built that incorporates our model of visual attention. Our system plays a video game in real time from a simulated retina; its input consists of a 500 by 500 array of binary pixels, which are processed at a speed of 40 frames per second. It achieves good performance while examining less than 1% of the pixels and 3% of the objects. Section 4 describes in some detail the key attentional mechanisms responsible for the system's success. In section 5 we attempt to analyze our system's performance, in order to further elucidate the role of attention. Finally in section 6 we compare our system with some related approaches, especially [5].

2 Why does visual attention matter?

The argument that focus of attention is important is not new. Blackboard systems [6, 7] for example have some notion of focus, as do production systems which model working memory [3], and even one automatic theorem prover [11]. The reason for studying focus is clear; in almost any complex domain, there will be a great many places where computational effort may be expended, and most of that effort must be put where it can yield a reward.

However, we believe that a specific sort of focussed attention will be necessary in order for autonomous systems to function in the real world. It is traditional to divide such a autonomous system into two components: the visual system, which describes the environment; and the central system, which determines what to do next. Work on focus in AI has concentrated on reasoning, but we believe that focus must play a key role in perception.

Our argument is based on both psychological and computational grounds. Psychological data suggests that biological visual systems employ selective attention, while computational considerations argue that attention should be part of perception.

We turn to the psychological argument first. The dominant paradigm in computer vision has been scene recognition (sometimes called "inverse optics"). The task of the visual system, in this view, is to describe all the objects in the scene in terms of their type, position and alignment. The visual system of a robot driving on the freeway, for example, would be expected to describe where all the currently visible cars are (not to mention the trees, signs, et cetera). However, there is considerable psychological evidence that biological visual systems do not work this way.

Psychological studies of attention like those summarized in [14] suggest that recognition is done only in a very small number of places. The studies suggest the existence of an internal attentional "spotlight", the locus of recognition, which can be moved around the image. Outside of the focus area(s) object recognition is not done, but certain events (often called "pop-outs") can be detected and located. An example of a pop-out is line orientation; a vertical line will pop out in a set of horizontal lines. Some properties which pop out are listed in [15].

If one were to design the visual system for a driving robot based on this psychological evidence, it would only perform recognition on a few objects at once, but would detect the presence of certain stimuli (such as a flashing blue light) anywhere in the image. This is exactly the kind of visual system that we propose.

Computational considerations also suggest such a model. For three reasons, it seems to us impractical for a robot's visual system to recognize all the objects in view at every time instant.

First, if all the objects were recognized all the time, the system's performance would suffer as the number of objects increased, even if almost none of the objects had anything to do with the robot's task. Second, the number of objects in a natural scene (such as a forest) is likely to be vast. If one could somehow recognize the objects at the appropriate level of abstraction (such as a pile of leaves), this might be feasible. Yet the appropriate abstraction is entirely task-specific, and trying to find this level of abstraction for each object hardly seems practical. Finally, if the visual system *were* to recognize all the objects and pass them to the central system, the central system would then face the difficult task of deciding which ones were relevant to the task at hand. This last point is of particular interest, because it highlights the difference between our notion of focus and the way that focus has been approached in previous systems.

In general, AI systems have been designed under the assumption that the visual system recognizes and locates all the objects in the scene, and the central system then reasons about these objects to determine what to do. With the exception of Agre and Chapman's work [2] (which we shall address later), these systems have assumed that the proper place for focus is in reasoning. A computational advantage of making focus part of perception is that the spatial structure of the environment has not yet been abstracted away, and can provide important clues in order to determine what to look at.

The spatial layout of the environment is important because the natural world has *locality.* By locality, we mean that effects in the world do not propagate instantaneously. As a consequence, objects that are near to you are inherently more likely to be interesting than objects further away. A nearby object can affect you immediately, while a far away object will not affect you until much later (and in general will need to come near first). In addition, objects that are near to other interesting objects are inherently interesting by locality, as the two objects can interact. The attentional mechanisms described in section 4 make extensive use of locality.

We therefore propose that the proper way to build autonomous systems that act in the world is to incorporate focus mechanisms into perception. We believe that these systems should only be able to look at a small subset of the available objects at the same time, although they should be able to detect pop-outs anywhere in the image.

One might well wonder whether it is possible to build a system which *does* anything based on such a visual system. A prerequisite for answering this question is a more detailed model of attention.

2.1 A model of attention

Our model of attention has top-down elements controlled by the central system, and bottom-up elements driven off the retina. There are two principle top-down components: a cache of recognized objects, and a spotlight for performing serial search on the retina. Pop-out features form the principle bottom-up element.

We assume a cache of fixed size that contains a small number of objects which can be tracked simultaneously. The type, position, alignment and velocity of each tracked object is also assumed to be available. In addition, we assume a visual “spotlight” which can examine small regions serially. The cache of tracked objects can be changed by pointing the spotlight and requesting that whatever object is at that point be added to the cache. (Of course, since there is a strict limit on the number of objects that can be tracked, this may cause some previously tracked object to become untracked.) We also assume that pop-outs can attract the spotlight to a point where the pop-out occurred. The central system can then decide if the object causing the pop-out should be tracked.

Our model is largely consistent with what is known about the human visual system, although the details of our proposal are mostly motivated by computational considerations. There is good evidence for the ability to track independently moving objects, although the precise number that can be tracked simultaneously is a subject of current research [13]. The idea of pop out features has been extensively studied by Treisman [15].

3 Flox: A system with selective attention

We have designed an architecture, Flox, that acts using our model of attention. We have implemented a version of Flox that plays a video game in real time on a Symbolics 3600. The particular video game we have chosen is Pengo.[1] Because we are interested in the issues surrounding perception, our program plays the game through a simulated retina (unlike the system described in [2], which had direct access to the internal data structures of the game). The video game produces a 500 by 500 bit-map approximately 40 times per second; for a

[1]David Chapman has kindly provided us with his implementation of Pengo.

human to play Pengo, the bitmap is displayed on the computer screen. Flox takes these 500 by 500 bit-maps as inputs and decides which action to take.

Our system plays Pengo moderately well, at a level roughly comparable to a human who has played for a few hours. Flox does this despite never looking at more than 7 objects at any one time (*which* objects it looks at, of course, varies as the game progresses). Furthermore, Flox takes 10,000,000 pixels per second as input, yet it only examines 30,000 of them (approximately 27,000 of these are examined by bottom-up vision, and hence could be processed in parallel).

3.1 The architecture

In this section we present the overall Flox architecture. First we introduce the notion of an *activity.* We then describe the flow of control within the architecture.

3.1.1 Activities, roles and casting

Flox assumes that there is a small number of goals that can be pursued, and further that these goals are largely independent of each other. With each goal we associate an *activity*, which attempts to achieve that goal. (This is similar to several common notions in AI, especially the agents proposed in the Society of Mind [12].) An activity consists of the following parts.

- A set of local variables (or *roles*),
- a function for evaluating a particular binding of these variables,
- a function that finds a new binding, and
- a function which suggests an action based on the current binding.

Each of these will be described in turn, and illustrated via the kill-monster activity for a video game.

Associated with each activity is a small set of *roles.* A role can be thought of as a variable which can be bound to a (tracked) object. An example might be the monster you are currently trying to kill (as part of the kill-monster activity). A binding of the roles in an activity will be referred to as a *cast.* The kill-monster activity could have as many different casts as there are monsters in the game.

An activity also has an *evaluation function*, which evaluates the activity's current cast. This evaluation function looks at the tracked objects that are currently bound to the roles, and determines how useful this binding is for achieving the activity's goal. The evaluation function for kill-monster might rate casts in which the monster is worth many points as preferable to those in which the monster is worth fewer.

An activity's *casting function* takes the current cast and tries to improve it (under the metric supplied by the evaluation function). The casting function can change the way that the tracked objects are assigned to roles, but, more importantly, it can also change which objects are tracked. The casting function and the evaluation function can be thought of as a generator and a tester, respectively, for casts. The kill-monster activity's casting function might use the spotlight to try to find a better monster to kill.

The last part of an activity is the *action function.* This function suggests a discrete action based on the current cast. The strength of this suggestion (which must compete with the alternative suggestions made by other activities) is the goodness of the current cast, as defined by the evaluation function. The action function for the kill-monster activity might suggest hurling an appropriate weapon at the monster.

3.1.2 Deciding what to do next

The basic operation of the system is to choose the activity with the best cast and perform the action it suggests. This decision is reconsidered on each cycle. Our system is in this sense "reactive".

While this is going on, each activity's casting function is trying to improve its current cast. Essentially, an attempt is made to hill-climb in the space of casts, where the goodness of a cast is defined by the activity's evaluation function. The casting mechanism is fairly complex, due to resource contention between different activities, and will be described in detail in section 4.1.

3.2 Visual system primitives

Implementing the Flox architecture requires a visual system that can do object recognition, object tracking, pop-out feature detection, and spotlight search. In this section we describe the operations our system performs on images to achieve this functionality. The nature of these operations is largely a product of the kind of inputs the system gets (i.e. bitmaps) and the kind of hardware the system runs on (i.e. a slow, serial computer).

The fact that Flox does not play Pengo using a camera, but receives copies of the video game display, allows object recognition and tracking to be performed very simply. All occurrences of a type of object on the image are identical.

3.2.1 Object recognition

On the Pengo video game display, black pixels are invariably part of objects, and hence are the key to object recognition. Given the bitmap coordinates of a particular black pixel, the system can determine the type and position of the object which contains the pixel. This recognition is computed directly from the image, without benefit of any intermediate representations (e.g., edges), using a large, machine generated decision tree. Recognizing an object takes about 1 millisecond.

3.2.2 Object Tracking

Flox maintains a small fixed-size (seven element) cache of tracked objects Each slot of the cache contains the type, position, and momentum (ie, the time and direction of last motion) of the corresponding object.

On every cycle of the game Flox attempts to re-recognize each object being tracked. This recognition task is much simpler than the general recognition task, as no object moves very far in one cycle. Re-recognition is performed using a smaller machine-generated recognition procedure and takes about .2 milliseconds.

3.2.3 Pop-out feature detection

The only pop-out feature used in Flox is the transition of a pixel from white to black. Such a transition can be produced by two kinds of changes on the video game screen: an object can move or a new object can appear.

A check for pixel transitions is made every now and then (approximately 6 times per second). Whenever a check is performed, the current display is copied onto an alternate bitmap. The next time the check is made, the black pixels on the current bitmap which were white on the previous bitmap constitute the regions of change.

3.2.4 The spotlight

The spotlight in Flox can be moved one pixel at a time across the image testing pixels. Determining whether one object will collide with another, for example, is done by scanning the intervening path for other objects. These operations take time proportional to the size of the area scanned.

4 Key attentional mechanisms

The primary purpose of attention is to select a reasonable cache of objects to track. The two processes which work toward this end are role casting and serial search for pop-out features.

4.1 Role casting

The role casting mechanism attempts to obtain the best possible cast for each activity (as determined by that activity's evaluation function). On each cycle, each activity's casting function tries to improve on its current cast, simultaneously.

If each activity's current cast can be improved without changing the cache of tracked objects or using the spotlight, there is no interaction between the activities. Each activity has a distinct set of roles, and the casts can be improved independently. Complications arise because the different activities must share the attentional resources of the visual system.

Each activity's casting function contends for places in the cache of tracked objects, and for the spotlight. The object cache has a fixed size, and different activities will in general have different opinions as to which object should be removed from the cache when it overflows. In addition, there is a single spotlight, and only a limited number of pixels can be examined in a single cycle.

Our system handles contention for the object cache by a weighted voting mechanism. Every activity is assigned a number of votes based on the goodness of its cast; these votes are evenly divided among the objects which are bound to the activity's roles. The total number of votes a tracked object receives (from all activities) determines its importance; when a place in the object cache must be freed, the least important object becomes untracked.

Contention for the object cache is also a source of serendipity, however, if we assume that the same type of object can be used in different activities. One activity can bring an object into the object cache, and that same object can be used by a different activity for another purpose.

Contention for the spotlight is handled differently. The casting functions which wish to use the spotlight are in general quite expensive. Typically they wish to search out in some direction for an object of a given type. We handle spotlight contention by occasionally allowing a casting function to use the spotlight. The casting function for an activity usually starts the spotlight on a tracked object and moves it about in some routine pattern. This process bears some resemblance to Ullman's notion of visual routines [16].

This description of casting is somewhat simplified in several respects. First, the spotlight can be used as part of action as well as part of casting, which is an additional source of contention. Second, certain activities require casts with some property that must be verified by the spotlight. For example, an activity might require two aligned objects with freespace between them. In this case the casting function would occasionally check that the current cast still has this property, instead of trying to find a better cast. Occasional checking constitutes implicit state, and suffices since the world does not change too quickly.

There are thus some additional cases of resource contention and serendipity. Nonetheless, the general picture of casting we have just provided is essentially accurate.

4.2 Serial search and inhibition of return

As a means of changing the cache of tracked objects, role casting has a very top-down, purpose specific nature. Pop-out features provide a bottom-up way of selecting areas of the retina (outside of the tracked objects) for possible tracking. Selected areas must still be serially examined to determine their importance. The only pop-out feature used by Flox is change on the retina.

In order to deal with areas containing pop-out events, the retina is partitioned into a grid of regions. This grid is searched for a region containing a pop-out event. The region containing the player's character is tested first. Next, the regions containing other tracked objects are examined; objects popping out in these regions are likely to be important due to their proximity to tracked objects (because of locality). Finally, the rest of the screen is searched outward from the player's character. When a pop-out feature resulting from an untracked object is detected, the object is identified and added to the cache of tracked objects and the search is halted.

This method of handling pop-outs would be nearly good enough, except when an object pops out that, upon closer examination, is found to be not worth tracking. If the object continues to pop out and the search order is the same, the object will be reacquired the next time a search for pop-outs is made, and will prevent any other pop-out object from being tracked. Tracking every object which popped out, or examining every pop-out event on the retina, would solve this problem. However performance would necessarily get worse as more pop-out events and objects appeared on the retina.

Another approach known as "inhibition of return" in the psychological literature [9], is to penalize regions which have been recently searched. In a video game, certain areas must always be tested. In particular, the regions adjacent to the character controlled by the player are always checked. It is usually acceptable to acquire a distant object a little late; however, nearby objects must be tracked immediately for survival (as a consequence of locality).

Our implementation of inhibition of return is based on the grid used for testing for pop-outs. Each region has a timestamp telling when it should next be searched. The search is as described above except that only areas that have not been looked at recently or are adjacent to a tracked object are considered.

Since only one new object is tracked when the system searches for pop-out events, pop-out objects will not necessarily be tracked immediately. However, if an object continues to pop-out it is likely that it will be tracked eventually, especially if it is near something interesting.

5 Analysis

How is it that Flox plays a video game from bitmaps in real time? What properties of the video game allow the system to succeed? Pengo exhibits spatial locality, and playing the game can be decomposed into a set of largely independent activities. Once good casts are available, choosing an action is straightforward, and good casts can be obtained by hill-climbing. Flox can act in real time because acting has been reduced to two cheap operations: selecting an action given a set of casts, and incrementally improving the casts.

5.1 Selecting an action given a cast

Given a cast for an activity in Pengo, the proper action can be read from the geometry of the objects bound to the roles. Consider Figure 1 as a simplified representation of the larva squishing activity. The figure depicts a larva, and an icecube against which to squish the larva. The larva squishing activity will attempt to move underneath the larva and squish it against the icecube above. The larva and the icecube are cast in the roles of the larva-to-squish and the icecube-to-squish-the-larva-against. The player's icon could be in any of the regions surrounding the larva and the cube. Each surrounding region is labeled with an arrow indicating the direction the player would go if it were in the region. In Pengo, the player could be in one of about 2^{19} distinct positions relative to the larva and icecube yet according to the diagram, the player need only distinguish seven action regions. Similarly, the goodness of a cast can be determined as a function of the geometric layout of the objects in the cast and the distances between them.

Table 1 indicates the simplicity of choosing actions given a cast for the five activities in the Flox system. The first row of the table indicates the number of roles in each activity. The second row presents the number of possible configurations after eliminating various symmetries, but including all differences in relative position. The third row is the number of regions that Flox distinguishes for choosing an action. The fourth row is the number of classes of configurations that Flox distinguishes for determining the goodness of a cast. Activity (E) is the (unsimplified) larva squishing activity diagrammed in Figure 1.

Thus given a set of casts, Flox need only read the goodness of each cast, and the suggested action for each activity from the image, to compute the next action.

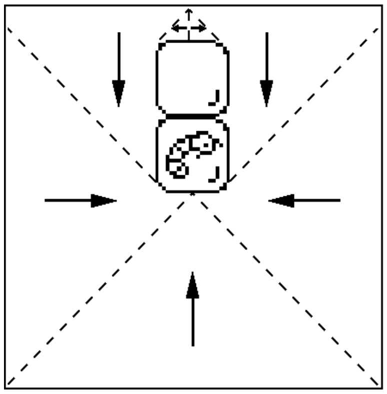

Figure 1: The simplified action function for squishing a larva

	Activities				
	A	B	C	D	E
Number of roles	2	2	2	3	3
Possible configurations	1000	1200	20	2^{31}	2^{19}
Action regions	9	3	6	12	9
Evaluation function regions	3	5	3	6	3

Table 1: Situations and their partitions

5.2 Choosing the right casts

The other half of playing Pengo is choosing the right casts for the activities. If the system were to analyze a new image on each cycle from scratch, it would be faced with any one of more than 2^{2000} possible game screens. Making sense of the game would be an expensive process. Flox makes controlled use of state to solve this dilemma. It takes advantage of the fact that it has a cache of tracked objects, and an assignment of these objects to roles.

With this limited state, Flox need only incrementally improve the current set of tracked objects and casts. The set of cached objects can be improved by searching for pop-out features. Both the cached objects and the set of casts can be improved by performing casting operations. Examining pop-out features and casting both involve expensive visual operations. However, the goodness of the cache of tracked objects and the set of casts is assumed to degrade slowly, and so these operations need only be done occasionally.

If there were no cache of tracked objects, or if good casts could be formed from widely separated objects, even a single casting operation could be prohibitively expensive. Principle elements of a cast, though, can usually be found in the cache, while other elements of the cast can be found by performing local visual routines starting with the principle elements.

An example of casting for the larva squishing activity in Flox will perhaps make this point clear. Suppose the the system has just rubbed out its current larva-to-squish. Suppose also that in the process of looking here and there, the system has started to track another nearby larva. Flox can immediately cast this larva as the larva-to-squish. All that remains to be done is to find an icecube-to-squish-the-larva-against. Flox serially examines a few pixels on each of the four sides of the larva. If no icecube is found the casting operation fails.

6 Related work

Agre and Chapman [2] built an earlier program which played Pengo. This program, called Pengi, is also discussed in Agre's thesis [1]. Chapman's recent work [5] describes a similar system called Sonja in much greater detail, and so we will concentrate on his work.

Our research has been heavily influenced by Pengi and Sonja. In particular, we have taken the idea of trying to use a model of attention as a basis for action. Our model of attention is similar to theirs; they also track a fixed number of objects and make use of visual routines.

However, Flox is significantly different from Sonja, because we have different research goals than Chapman. We are interested in studying the computational requirements for systems that act in the world. Chapman is largely interested in implementing psychological models of attention.

The first difference lies in the fact that we take bitmaps as input, while Sonja accesses the internal data structures of the video game it plays. Using bitmap input has several consequences.

- Since object recognition is moderately expensive, and visual search takes time proportional to the area processed, Flox must limit its use of visual operations according to these real costs. Sonja, on the other hand, is limited by psychological constraints and the implementation dependent costs of accessing the video game's internal data structures.

- Because the spatial structure of the world is reflected in the bitmap, it is easy to take advantage of serendipity. When Flox places the attentional spotlight on an object and moves it across the bitmap, it can encounter other nearby objects along the way, and use these serendipitous events to advantage.

- It would be possible to hook Flox up to a genuine camera without massive amounts of additional research. All that would be required is a connection machine implementation of the primitives described in section 3.2.

It suffices for Chapman's purposes to simulate human low-level vision. Our interest in the computational constraints on action, on the other hand, forces us to use retinotopic input.

The other major difference is modularity. Our system provides a domain-independent control mechanism which includes the attentional apparatus. This control mechanism allows for a modular decomposition of the task into separable activities, and a decomposition of each activity as described in section 3.1.1. Sonja has no such decomposition, and consists mostly of complex special purpose code. As a result, the system is difficult to modify (for, e.g., learning). One example of the difference in modularity is the way the two systems decide what to do next. Sonja has a monolithic special-purpose function which must balance all considerations to compute a good action in any situation. Flox decomposes the task into separate action and appropriateness functions for each activity and combines their results using a voting scheme to choose an action. Modularity, of course, is not a requirement for Chapman, because of the goals of his research.

The control mechanism used by Flox has a certain similarity to blackboard architectures such as [6]. The cache of tracked objects can be viewed as a blackboard, in that it is a centralized resource through which the various activities communicate. However, we are interested in different sorts of domains, and hence different issues, than those typically addressed by blackboard research community.

There are many other systems which attempt to achieve real-time response in complex domains by using something like boolean circuitry. Kaelbling's work on GAPPS [8] is one such effort, and similar work has been done by Maes, Nilsson and Schoppers, among others. These systems have not so far been hooked up to a camera, or to a simulated retina; as a result, they have not had to address the problem of selective attention.

The control flow in the system designed by Maes [10] has some definite similarities to our scheme. Control in her system does not result from some prewired monolithic procedure, and can thus emerge from the interaction among different goal-pursuing 'agents'. However, because she does not model perception, her notion of goals is very different from ours. A single goal (i.e., killing a monster) has many different possible instantiations (i.e., one per monster). Controlling such instantiations is a major motivation for studying selective attention, and is a key task handled by Flox.

7 Conclusions

We have examined the issue of visual attention, and have shown that a model of attention can provide a basis for action. We hope that this research will prove useful as the AI community moves towards autonomous systems with realistic perceptual systems.

Acknowledgements

We wish to thank John Batali, David Chapman, Dan Huttenlocher, Leslie Pack Kaelbling, Jim Mahoney, Jeff Shrager, Devika Subramanian and Anne Treisman. John Woodfill is supported by a fellowship from the Shell corporation. Ramin Zabih is supported by a fellowship from the Fannie and John Hertz foundation. Summer support was provided by Xerox Corporation and by ONR contract N00014-81-K-0004.

References

[1] Philip Agre. The dynamic structure of everyday life. Technical Report 1085, MIT AI Lab, October 1988.

[2] Philip Agre and David Chapman. Pengi: An implementation of a theory of activity. *Proceedings of AAAI-87, Seattle, WA*, pages 268–272, July 1987.

[3] John Anderson. *The Architecture of Cognition.* Harvard, 1983.

[4] Rod Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, March 1986.

[5] David Chapman. Instruction use in situated activity. Technical Report 1204, MIT AI Lab, February 1990.

[6] Les Erman, Frederick Hayes-Roth, Vic Lesser, and Raj Reddy. The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12:213–253, 1980.

[7] Barbara Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–322, July 1985.

[8] Leslie Pack Kaelbling. Goals as parallel program specifications. *Proceedings of AAAI-88, St. Paul, MN*, pages 60–65, August 1988.

[9] Raymond Klein. Inhibitory tagging system facilitates visual search. *Nature*, pages 430–431, August 1988.

[10] Pattie Maes. The dynamics of action selection. *Proceedings of IJCAI-89, Detroit, MI*, pages 991–997, 1989.

[11] David McAllester. *ONTIC: A Knowledge Representation System for Mathematics.* MIT Press, 1989.

[12] Marvin Minsky. *The Society of Mind*. Simon and Schuster, 1986.

[13] Zenon Pylyshyn. The role of location indexes in spatial perception: A sketch of the FINST spatial-index model. *Cognition*, 32(1):65–96, June 1989.

[14] Anne Treisman. Features and objects in visual processing. *Scientific American*, November 1986.

[15] Anne Treisman and Stephen Gormican. Feature analysis in early vision: Evidence from search asymmetries. *Psychological Review*, 95(1):15–48, 1988.

[16] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.