Visual routines and visual search: a real-time implementation and an automata-theoretic analysis

Ian Horswill
MIT Artificial Intelligent Laboratory
545 Technology Square
Cambridge, MA 02139 USA
ian@ai.mit.edu

Abstract

I describe a real-time implementation of Ullman's visual routine processor (VRP) theory of intermediate vision for visual search. The system performs serial self-terminating visual search and computes 2D spatial relations of objects from live color video using low cost hardware. I present a formal model of a VRP with unbounded resources and quantify the amount of external control structure required to solve Horn clauses using the VRP. In discussing the effect of resource limitations I show that contemporary models of biological visual attention are unable to solve surprisingly simple queries. I also describe a novel logic programming system that finds satisfying variable assignments for Horn clause queries using the VRP. The system contains no internal database: all logic variables are directly grounded in the world using VRP queries. Finally, I briefly discuss experiments with natural language interpretation and motor control using the VRP. Experiments on real data are given.[1]

1 Introduction

Shimon Ullman proposed the visual routines theory of intermediate vision as a way of explaining how the human visual system might solve certain visual tasks (such as computing spatial relations) that seem to require serial processing [Ullman, 1984]. At a gross level, the theory proposes that the visual system contains a set of registers that can contain different types of visual data, a means of focusing visual attention on task-relevant portions of the image, and a set of primitive "instructions," such as coloring and line drawing, that can be combined like instructions in a computer program to compute useful properties of an image. These resources are collectively referred to as the *visual routine processor* or VRP.

The visual routines theory have received increasing attention from the AI community in recent years ([Agre and Chapman, 1987] [Chapman, 1990][Reece and Shafer, 199l][Romanycia, 1987][Whitehead and Ballard, 1990]). Much of this attention has come from the reactive reasoning and planning community, in part because Agre and Chapman's implementation of the visual routines model provides an alternative interface between reasoning and perception, which is both easier to implement and more biologically plausible than standard database-like interfaces.

Despite the level of interest, there has yet to be a VRP implementation that runs on real camera images. To date, VRP systems have run either off of hand-drawn bitmaps [Romanycia, 1987] or have been directly interfaced to the world model of a world simulator, thus bypassing low-level vision entirely [Agre and Chapman, 1987][Chapman, 1990][R eece and Shafer, 1991].

In this paper, I describe Jeeves , a working visual routine processor that runs off of color video at approximately 10Hz using only relatively simple hardware. This is work in progress. As of this writing, Jeeves implements the visual search portion of the theory, the part which is presently best worked out. I will also present a formal analysis of the amount of control machinery required to solve arbitrary-length Horn clauses, discuss the effect of resource limitations on visual search, and give an example of a simple Horn clause for which most current biological theories of visual search cannot account. Finally, I will describe Bertrand, a novel logic programming system that answers Horn Clause queries about scenes without the use of an internal proposition database.

2 Overview of the VRP model

2.1 Early vision and attention

Virtually all theories of vision presuppose a stage of *early* or *low level* processing that extracts local features from the image such as edges, color information, or depth. It is generally believed that early vision computes a set of *low level maps* each displaying the strength or value of a specific feature for all points in the image. These maps are believed to be computed bottom-up and in parallel.

Support fot this research was provided in part by the University Research Initiative under Office of Naval Research contract NO0014-86-K-O685, and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-85-K-0124, and in part by the National Science Foundation under grant number IRI-9407273. Joanna Bryson kindly read drafts of this paper and provided useful comments.

In the last decade, covert visual attention, the problem of selecting image-plane regions for processing, has been widely studied in the psychophysical and neurophysiological communities. A wide range of neural models of covert attention have been proposed (see [Tsotsos *et* a/., 1994] for a detailed survey). These models assume image regions are selected based on their low level features (see above). Most use a pyramid structure, and so are referred to as "attention-" or "addressing-pyramids". Models differ on what the attention mechanism reports to the next level about the region. Some models such as [Koch and Ullman, 1985] route aggregate feature values of the attended region to their outputs, while other models, such as [Olshausen *et* a/., 1992], route resampled images to their outputs. Jeeves , is patterned after [Koch and Ullman, 1985J, although dynamic resampling would be easy to add.

For the purposes of this paper, the most important data on covert attention are the experiments of Treisman *et al.* [Treisman and Gelade, 1980], which suggest the vision system can search all points in the image in parallel for many low level features (called "pop-up" properties) but must handle conjunctions of features by serially enumerating regions satisfying one of the conjuncts and then testing them for the other(s). Most neural models of attention have image-parallel hardware for matching pop-up properties to handle the simple cases, and provide some *return-inhibition* mechanism to support region enumeration for conjunctions. Return-inhibition stores the locations of previously attended regions and prevents them from being selected in the future. All regions satisfying a given pop-up feature can then be enumerated by repeatedly selecting a region and inhibiting it.

It is outside the scope of this paper to debate the validity of Treisman's experiments or the various neural models of covert attention. The interested reader is directed to [Tsotsos *et* o/., 1994].

2.2 Visual routines

The visual routine model [Ullman, 1984] claims that certain kinds of visual work are done by selecting relevant regions of the image and applying simple geometric operations to them such as drawing lines to connect them, searching along the lines for other regions, or checking whether a point lies within a closed curve using flood-fill operations. The essential claims are that (1) there exists a *visual routine processor* (VRP) that consists of a set of discrete functional units, each capable of performing a specific operation such as line drawing or flood filling and (2) these operations can be combined in task-specific manners to do useful visual work in the same way subroutines are built up from primitive machine instructions (hence the name "VRP"). The model presupposes a number of basic architectural features of the visual system:

- An early vision system to compute primitive features,
- An attention system to select task-relevant regions based on those features and route them to the VRP,
- A set of functional units for executing primitive instructions, and

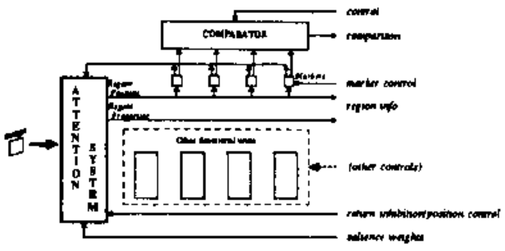

Figure 1: Visual routine processor architecture.

- A set of specialized state elements for holding intermediate values (registers).

The details of the registers and functional units are unknown. Chapman's system [Chapman, 1990] contained four types of state elements: markers, which held image locations; lines and rays; activation planes, which were general registers that could hold arbitrary binary images; and a return inhibition map. All state elements except the return inhibition map could be directly named and manipulated. The return inhibition map had its own specialized operations. In Chapman's system, a blocks-world visual routine for finding the first block underneath some blue block would consist of the following operations:

1. clear the return inhibition buffer
2. select a blue block and inhibit it
3. set marker 0 to the location of the selected region
4. set ray 0 to project downward from marker 0
5. set marker 1 to the first object along ray 0
6. if the previous operation failed, go to step 2.

If this routine succeeds, then marker 0 will designate the blue block and marker 1 the block underneath it.

3 Jeeves implementation

Jeeves is the first system to implement the VRP model on real camera data using a real low level vision system. It implements the necessary extensions to allow operation on scenes in which objects have non-trivial spatial extent and/or occlude one another. It implements marker and line operations, but presently does not implement activation plane operations (the current domain doesn't exercise them adequately). It implements queries on 2D spatial relations and simple feature values and has been tested on children's blocks (real ones, with differing shapes), books, and brightly colored kitchen utensils. The architecture of Jeeves is shown in figure 1. I hope to apply it to other domains and to extend its repertoire of computations soon.

Jeeves runs on 64 x 29 images at approximately 10Hz. It is written in C and presently runs on color vision hardware designed at MIT by Chris Barnhart. The hardware consists of roughly $1000US worth of components, including a Texas Instruments C31 floating-point DSP and an NTSC color digitizer. It is controlled from a Sparc-1 running Scheme over a 38K-baud serial port.

The process of building a low-level vision system and VRP that can run at 10Hz required a great many compromises, most importantly, the use of low resolution images and the lack of a shape-matching system. In some cases, these compromises limited the tasks that could be solved by the system. I have tried to restrict my attention to claims and tasks that are unaffected by these limitations.

3.1 Low-level maps

After grabbing the image and averaging it down to low resolution, the system computes a number of low level retinotopic maps. As of this writing, it computes various color maps (R, G, B, R/I, G/I, B/I), intensity, temporal and spatial derivatives, including Laplacian, $\nabla^2 G$ edges, and a grey-scale symmetry operator. An optic flow detector has also been implemented but is not presently used.

3.2 Preattentive segmentation

Chapman's SIVS system ignored the issue of dividing the image into regions corresponding to distinct objects, effectively assuming that objects were single points. Chapman argued from psychophysical evidence that the vision system might perform a first-pass segmentation before the attention pyramid. The attention pyramid would then select among segments rather than points. Other models assume that the attention pyramid can select among different scales, but does not use segmented input. As much for engineering reasons as for theoretical ones, I choose to use a preattentive color segmentation system as a preprocessing stage for the attention pyramid. The system finds color boundaries in the image and uses a connected-components algorithm to label regions of homogeneous color. The segmentation system also allows the higher level to specify minimum and maximum segment sizes. Minimum sizes allow the suppression of "noise" segments and maximum sizes act as a crude form of figure/ground separation.

I do not propose the simple color segmentation system as any kind of theory of human preattentive segmentation. However, the use of segmentation is logically independent of other design decisions; one could change or remove the segmentation system and still have a usable vision system.

3.3 The saliency map and visual attention

The VRP selects task-relevant image regions by computing a pixel-by-pixel weighted sum of the low level maps [Clark and Ferrier, 1988][Ahmad and Omohundro, 1991]. The weights are one of the input parameters of the system and can be changed continuously by the higher levels (see figure 2). Weighted sums may be more powerful than the human system. One can adjust the weights to select for conjunctions, for example. However, this would presuppose something like a perceptron learning algorithm running to learn the proper weights for the proper conjunctions. Since this would require considerable training, the human system might use the weighted combinations and still be unable to do parallel search for conjunctions. In point of fact, more recent psychophysical results do indicate that at least some conjunctions

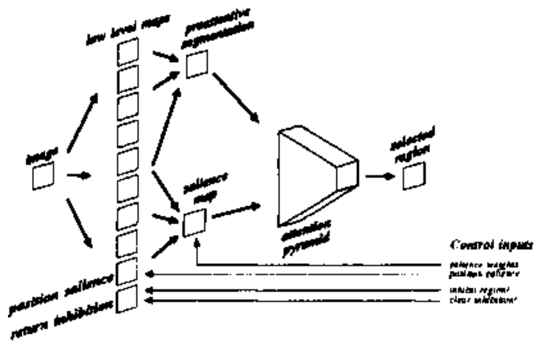

Figure 2: Structure of the attention system

can be learned with sufficient training (see [Weismeyer, 1992] and [Tsotsos *et al.*, 1994] for surveys of recent results).

The attention system computes the region with the maximum integral of salience. This differs from previous systems which have either found the image point with maximal salience or the image point with maximal smoothed salience, perhaps over several different scales. Finally, the attention system computes the bounding box, area, centroid, and average low level map values for the winning segment.

3.4 Visual markers

The only directly-addressable memory in Jeeves is a small collection (10) of registers called "markers" [Chapman, 1990] which can hold the centroid and bounding box of a segment. Markers can be loaded with the specifications of the currently attended region, compared to one another, or used to focus attention (see below).

3.5 Return-inhibition and spatial constraint

In many biological attention models and also in many computer implementations, return inhibition is implemented as a function internal to the attention pyramid itself. Indeed, most biological models provide only very limited control over return inhibition, if any. Chapman's model provides control lines to enable and disable the addition of the current point to the set of inhibited points and to clear the set of inhibited points. It does not provide the capability to suppress return-inhibition without permanently clearing the set of inhibited points.

Jeeves implements return inhibition through a separate return-inhibition map (RIM) that is summed into the saliency map (see figure 2). Jeeves provides separate control lines for adding the current region to the RIM, clearing the RIM, and including the RIM in the the saliency measure. The latter is controlled through the same weighting mechanism used for other maps.

Both Chapman and Ullman propose that the VRP can search for objects along specified rays. In Chapman's system, this is implemented though a separate attention mechanism. As with return inhibition, spatial constraint of search is implemented in Jeeves through the saliency mechanism. A separate low level map, with its own independent weight, computes a "position salience" for each point in the image. Position salience is *task dependent* but *image independent* It is computed as a sum of terms that fall off quadratically with distance from a marker and from a ray emanating from the marker. If either term is zero, the position salience is zero. The marker, ray direction, and rates of decay are inputs to the attention mechanism.

4 Automata-theoretic analysis

Most computational work on covert attention and visual search *(e.g.* [Chapman, 1990][Tsotsos *et a/.*, 1994][Weismeyer, 1992]) has been designed specifically to fit psychophysical timing data on very specific tasks, such as Triesman's. Relatively little attention has been given to the computational power of these models in the abstract. In effect, these models treat the visual search system as a very simple Prolog engine: they search for image regions satisfying a conjunction of primitive features. They serially enumerate regions satisfying the first feature and test each for the other features. A failure of the test causes *backtracking,* which is implemented by the return inhibition mechanism. The essential differences between these systems and a Prolog engine are that:

- The propositional database is replaced by the image.
- Logic variables are replaced by markers or the attention output.
- Backtracking state is stored in the return inhibition mechanism, and
- Only very simple queries are examined.

The last two of these are particularly important for our purposes: the Treisman experiments amount to Horn clause queries containing exactly one variable. The visual attention mechanisms can only handle queries involving one variable, because they only have one state element in which to record backtracking information: the return inhibition mechanism. Multiple-variable queries require either multiple return-inhibition maps or *ad-hoc* auerv restrictions to limit backtracking.

4.1 Enumeration oracles

We can make these observations more precise by introducing an abstract VRP with infinite resources. First, some definitions. Consider a conjunction of literals in Prolog notation:

$$green(X), vertical(X), on(X,Y). \qquad (1)$$

We define the *signature* $P(f_1, ..., f_n)$, $f_i \in \{0,1\}$ of a predicate application $P(X_1, ..., X_n)$ to be the application with its arguments replaced by 1 if the application is the leftmost occurrence of the variable in the clause, else 0:

$$green(1), vertical(0), on(0,1)$$

The signature specifies whether the search engine needs to enumerate values for the variable (1) or to filter them (0).

We will define an *enumeration oracle* over a domain D to be a machine with an input (to be described momentarily), a binary output, and infinite collections of numbered *object registers* $O_1, O_2, ...$ ranging over D and *context registers* $C_1, C_2, ...$ ranging over sequences of tuples over D, *i.e.* $(D^*)^*$. Note that these machines are *oracles*: idealizations not intended to be physically instantiable.

The input to the oracle is a tuple $(S, (o_1, ..., o_n), c, r)$ where S is a predicate signature, $(o_1, ..., o_n)$ is a list of object register numbers (one for each argument of the predicate), c is a context register number, and r is a binary restart flag.

Upon receiving an input, $(P(f_1, ..., f_n), (o_1, ..., o_n), c, r)$, the oracle tests whether $P(O_{o_1}, ..., O_{o_n})$ is satisfiable, updates context register C_c and those object registers specified by the f_i, and returns true if the predicate was indeed satisfiable.

More formally, let v be the number of registers to be instantiated, *i.e.* $\sum f_i$. If $v = 0$, the oracle simply tests whether $P(O_{o_1}, ..., O_{o_n})$ is true. If $v \neq 0$, the oracle must determine if there is another set of variable values that will make $P(O_{o_1}, ..., O_{o_n})$ satisfiable. To do this, it maintains the set of untried variable values in context register C_c. If r is true, we are restarting an enumeration so the oracle chooses an arbitrary ordering on D^v and stores it in C_c. Regardless of r, it finds the first tuple in C_c that can be substituted into the unbound variables in the literal so as to make it true. If such a tuple exists, the O_{o_i} for which $f_i = 1$ are updated from the tuple, C_c is set to the remaining tuples, and the oracle returns true. Otherwise it returns false.

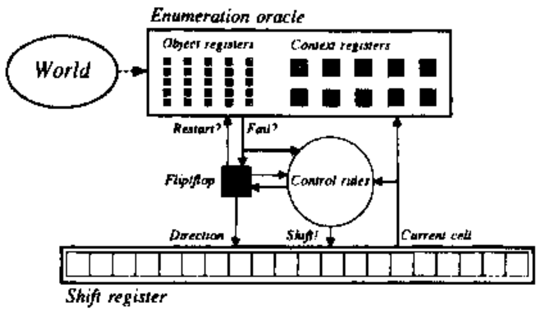

Figure 3: Solving Horn clauses with an enumeration oracle.

4.2 Solving Horn clauses

The enumeration oracle is essentially a VRP with infinite markers (object registers) and return inhibition buffers (context registers). It can be shown that such a system can answer arbitrarily long Horn clauses, given surprisingly little external control logic: a flip flop and a shift register (see figure 3). Because of space limitations, I will only sketch the proof.

The system requires an external "compilation" phase to translate from the external representation of the Horn clause to a suitable shift-register tape. For convenience, we will assume that variables are drawn from a numbered set X_i, which we will put in correspondence with the object registers O_i. Given a conjunction of literals over variables,

$$P_1(X_{i_{11}}, \ldots, X_{i_{1c_1}}), \ldots, \; P_n(X_{i_{n1}}, \ldots, X_{i_{nc_n}})$$

we create a shift register tape whose cells are oracle inputs that enumerate the respective literals of the conjunction. Thus cell i, is the tuple giving the signature of the ith literal, the registers for the literal's argument variables, and a context register to use. We will use context register c_i for literal *i.* The restart flag must be generated at run time, so it will not be included in the cell. The shift register tape is then:

$$((S_1, (i_{11}, \ldots, i_{1c_1}), 1), \ldots, (S_n, (i_{n1}, \ldots, i_{nc_n}), n))$$

and the tape for the clause in (1) would be (using *X* as variable 1 and *Y* as variable 2):

$$((green(1), (1), 1), \; (vertical(0), (1), 2), \; (on(0, 1), (1, 2), 3))$$

The only non-trivial part of this compilation process is computing the signature. It is easily done by keeping track of which variables have appeared so far in a left-to-right reading of the clause.

Given this tape, we can enumerate all satisfying variable assignments of the clause by adjoining a simple automaton to the oracle. The tape is augmented with left- and right- end-of-tape markers, and the shift register is started at the left end of tape. The automaton has a single flip-flop, called *restart,* which is initialized to 1. It follows the rules:

- If *restart,* then shift to the next cell to the right, else to the left.
- If not at end of tape, take current cell's tuple, append *restart* to it, and feed it to the oracle, storing the result in *restart.*
- If at the right end of tape, signal success and clear restart.
- If at the left end of tape and restart is clear, terminate.

These rules are easily implemented in a few gates.

In effect the tape acts as the backtracking stack, one cell per frame, and the context registers named on the tape hold the backtracking history of their respective frames. Each time the automaton reaches the right EOT and signals success, it has found a complete satisfying variable assignment for the clause. It can be shown that:

Claim 1 *The automaton above signals success once for each satisfying variable assignment of the clause.*

Sketch of proof: *By induction, the number of rightward tape crossings out of the ith cell is equal to the number of satisfying variable assignments for the first i literals in the clause. Rightward crossings out of the last cell enter the right EOT and so signal success.*

This allows us to quantify the amount of control state needed to satisfy arbitrary length Horn clauses with an enumeration oracle.

We have assumed the enumeration oracle handles arbitrary signatures of arbitrary literals. We can weaken these assumptions somewhat through compile-time transformations by forcing variables to be enumerated one at a time using weaker predicates or by evaluating the literals in a different order. Such transformations are outside the scope of this paper.

4.3 Resource constraints

Current biological theories of covert visual attention tacitly assume only a single "context register," the return-inhibition state. The return-inhibition state can only track the enumeration of a single variable. Thus, it would seem that the VRP can only handle clauses containing a single variable. In point of fact, this can be optimized somewhat.

Backtracking with markers

The predicate *on(X, Y)* be solved in the VRP by projecting a short ray downward from a known *X* and placing *Y* on the first object found along the ray (or by projecting upward from a known *Y).* The same procedure also works for the immediately-above *(iabove)* relation[2] if we use an infinitely long ray. A solution to *iabove(X,Y)* is also a solution to *above(X,Y).* To find any other solutions to *above(X, Y),* we can inhibit return to the previous solution and resolve. This requires an extra return-inhibition map, however. If we adopt the idealization that an object can only immediately above one other object, then we can solve *above(X,Y)* without an extra return-inhibitionmap. The reasin is that aboveness is the transitive closure of immediately aboveness:

$$\begin{aligned} iabove(X, Y) &\Rightarrow above(X, Y) \\ iabove(X, Z) \wedge above(Z, Y) &\Rightarrow above(X, Y) \end{aligned}$$

which gives us the following procedure for enumerating values of *Y* given *X:* to restart an enumeration, project a ray down from *X* and bind *Y* to the first object found along the ray; to backtrack the enumeration, project a ray down from the current value of *Y* and rebind *Y* to the first object found along the ray.

In effect, this technique causes the object registers to do double-duty as context registers. The technique is applicable to any right-unique relation:[3]

Claim 2 *Let $r(X, Y)$ be right-unique and $r^*(X, Y)$ be its transitive closure. An enumeration oracle can enumerate the signature $r^*(0, 1)$ without using a context register, provided that it can enumerate $r(0, 1)$.*

Method:
Given the input $(r^*(0,1), (o_i, o_j), c, 1)$, the oracle computes $(r(0,1), (o_i, o_j), c, 1)$. Given $(r^*(0,1), (o_i, o_j), c, 0)$, it computes $(r^*(0,1), (o_j, o_j), c, 1)$. Neither operation requires changing the context register *c*, so the oracle can safely ignore it.

Although useful, the technique does not obviate the need for context registers. Arches, a favorite topic in the blocks world, presuppose the *on* predicate is non-right-unique. An arch query such as:

$$on(X, Y), on(X, Z), Y \neq Z$$

simply cannot be processed by a VRP unless it has multiple return-inhibition maps! Of course, arches might be recognisable via a *different* query, depending on the particular capabilities of the VRP.

[2] *X* is immediately above *Y* iff *X* is above *Y* and for all *Z* above *Y* either *Z* is above *X* or $Z = X$.

[3] r is right-unique if for all x, y, z, $r(x, y) \wedge r(x, z) \Rightarrow r(z, x)$

```
(satisfy '((blue x) (above x y) (above y z)))
Down to 0   ; (blue x): x set to block 1
 Down to 1  ; (above x y): y set to block 2
  Down to 2 ; (above y z): query fails
 Up to 1    ; (above x y): query fails
Up to 0     ; (blue x): x set to block 3
 Down to 1  ; (above x y): y set to block 4
  Down to 2 ; (above y s): z set to block 5
7 VRP operations; The formula is satisfied.
```

Figure 4: An image (above left), its subsampled version (above right), and an example query and execution trace (below). Semicolons mark annotations. Note that the system assumes any uniformly colored region is a block. It does not recognize the other objects, although it can tolerate their presence.

5 A database-free logic programming system

Bertrand is logic programming system built to experiment with the search capabilities of the VRP. It is essentially a direct implementation of the Horn-clause satisfier discussed above. It has a compilation phase that translates clauses into shift-register tapes and an execution phase in which VRP markers are set to a satisfying variable assignment for the clause using backtracking search. Marker positions are overlaid as colored X's on the live video image. An example execution trace is shown in figure 4.

At present, Bertrand runs with only one return-inhibition map, although this could be easily changed.

6 Answering natural language queries

The above results suggest a natural extention to the processing of simple natural language utterances: we add a simple parsing front end and a semantics system that defines the semantics of visual words in terms of VRP instructions that find their referrents. We can then answer the sort of natural language queries whose logical forms are simple Horn clauses of the form that Bertrand can solve:

- Is there a green block?
- Is the green block on another block?
- Show me the blue block on the red block,
- Is it on a red block?
- Is there a green block on a blue block on a red block on an orange block under a green block?

Ludwig is a natural language system that can answer simple questions using the VRP. Although quite limited, it is novel in that it uses not only no world-model, but no tree-structured representations whatsoever! Ludwig is implemented entirely as a set of piplined parallel processes, each finite state, with fixed connections between them. Connections are simple activation levels, binary values, or narrow busses.[4] In this sense, it is compatible with connectionist ideas. To my knowledge, Ludwig is the most sophisticated system to date based on the notion of using the world as its own best model. It is also the only such system that supports fully compositional representation and recursive structures.

7 Motor control

We have mounted the vision system on a mobile robot base and used it for simple visuo-motor experiments. Although designed for visual search, the visual routine processor already contains much of the machinery necessary for the visual system of the Polly robot [Horswill, 1993]. For example, the Polly system performs collision avoidance by steering to avoid texture in the image. Since the floor of Polly's environment has no surface markings, the presence of an edge necessarily indicates the presence of an obstacle and so an edge detector is sufficient to act as an obstacle detector. Polly computes the depths of edges using image-plane height. In particular, Polly steers by computing three distances, d_j, d_c, and d_r,, which are defined as the image plane heights of the lowest edge pixels in the left, center and right thirds of the image, respectively. Polly then turns at a rate proportional to $d_l - d_r$ and advances at a rate proportional to d_c.

Such a control regime is easily added to the VRP. The output of the saliency map is fed to a motor control unit that computes d_l, d_c, and d_r, defined to be the image plane heights of the lowest *salient* pixels in their respective image areas. The motor control unit also computes z_{min}, the *x* coordinate of the lowest salient pixel in the whole image. A *motor control instruction* then consists of a VRP instruction to determine the settings of the salience parameters and a matrix *A* and a vector *b* used to compute the motor velocities as:

$$\begin{bmatrix} v_{rot} \\ v_{trans} \end{bmatrix} = A \begin{bmatrix} d_l \\ d_c \\ d_r \\ z_{min} \end{bmatrix} + b$$

For example, using a VRP instruction to attend to spatial derivatives and specifying

$$A = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ d_{stop} \end{bmatrix}$$

where d_{stop} is the desired stopping distance for the robot, yeilds the standard Polly collision avoidance algorithm, while the values

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} -d_w \\ d_{stop} \end{bmatrix}$$

where d_w, is the desired distance reading for the left wall, yeilds Polly's left-wall following algorithm. Finally,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ d_{stop} \end{bmatrix}$$

[4]In fact, Ludwig is implemented directly in a register-transfer-level hardware language.

yields a primitive following algorithm that steers toward and approaches the nearest object.

The motor control unit is also able to make ballistic turns to face in the direction of a given marker. After a ballistic turn of the robot's body, marker positions are automatically updated to compensate for the motion.

Finally, the motor control unit has been integrated into Bertrand to allow motor control operations to be mixed with visual search operations. Thus the program:

red(X), on(X, Y), blue(Y), face(X), approach().

will cause the robot to find a red block on a blue block and drive up to it.

8 Conclusions

This paper provides the first VRP implementation that runs on real camera data and one of the few implementations of biological theories of covert visual attention that runs on real data. It also provides a formal study of the VRP's visual search capabilities and the effects of the resource limitations in biological attention theories (for a study of visual search independent of the VRP architecture, see [Tsotsos, 1990]).

One of the outcomes of the formal analysis is that some surprisingly simple queries cannot be solved without assuming more machinery than current biological theories provide. This is not fatal to the theories, it simply means that must either (1) we revise the theory, (2) assume that some other system is in charge of finding arches, or (3) assume the problematic queries aren't important in everyday life. (1) is simple: just assume there are multiple return-inhibition maps and that their effects on the attention pyramid can be gated by higher level processes. This may not be true, but it suggests a whole range of interesting psychophysical experiments. (2) is more tricky. Many computer models of recognition tacitly assume the ability to perform complicated backtracking searches. Explaining one system's inability to backtrack by assuming another's ability to backtrack simply begs the question. If psychophysical experiments should show that the human visual system really is limited in its ability to backtrack in visual search, this would lend support to view-based models of recognition.

Bertrand and Ludwig are interesting because they have no internal world model: database operations on the model are translated directly into image-plane operations, yielding a high performance interface that is plug-compatible with a database. An existing problem solver could be modified to do the same. The result would be a system that seamlessly alternated between image operations and world-model operations, depending on the type of query.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence,* pages 268-272, 1987.

[Ahmad and Omohundro, 1991] Subutai Ahmad and Stephen Omohundro. Efficient visual search: A connectionist solution. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society,* Chicago, IL, 1991.

[Chapman, 1990] David Chapman. Vision, instruction, and action. Technical Report 1204, Massachusetts Institute of Technology, Artificial Intelligence Lab, April 1990.

[Clark and Ferrier, 1988] James J. Clark and Nicola Ferrier. Modal control of an attentive vision system. In *Proceedings of the Second International Conference on Computer Vision,* pages 514-523, Tampa, Florida, December 1988. IEEE Computer Society.

[Horswill, 1993] Ian Horswill. Polly: A vision-based artificial agent. In *Proceedings of the Eleventh National Conference on Artificial Intelligence,* pages 824-829. AAAI, MIT Press, 1993.

[Koch and Ullman, 1985] C. Koch and S. Ullman. Shifts in selective visual attention: Towards the underlying neural circuitry. *Human Neurobiology,* 2:219-227, 1985.

[Olshausen *et* a/., 1992] Bruno Olshausen, Charles Anderson, and David Van Essen. A neural model of visual attention and invariant pattern recognition. CNS Memo 18, CalTech Computation and Neuural Systems Program, Pasadena, CA, 1992.

[Reece and Shafer, 1991] Douglas A. Reece and Steven Shafer. Using active vision to simplify perception for robot driving. CMU-CS 91-199, Carnegie-Mellon University Computer Science Department, 1991.

[Romanycia, 1987] Marc H. J. Romanycia. The design and control of visual routines for the computation of simple geometric properties and relations. Technical Report 87-34, University of British Columbia, Vancouver, BC, Canada V6T 1W5, October 1987.

[Treisman and Gelade, 1980] Anne M. Treisman and Garry Gelade. A feature integration theory of attention. *Cognitive Psychology,* 12:97-136, 1980.

[Tsotsos *et* a/., 1994] John K. Tsotsos, Seam M. Culhane, Winky Yan Kei Wai, Yuzhong Lai, Neal Davis, and Fernando Nuflo. Modeling visual attention via selective tuning. Technical report, University of Toronto Department of Computer Science, 1994.

[Tsotsos, 1990] John K. Tsotsos. Analyzing vision at the complexity level. *Behavioral and Brain Sciences,* 13(3):423-469, 1990.

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

[Weismeyer, 1992] Mark David Weismeyer. An operator-based model of human covert visual attention. CSE-TR 123-92, University of Michigan Computer Science and Engineering Division, Ann Arbor, MI, 1992.

[Whitehead and Ballard, 1990] S. Whitehead and D. Ballard. Active perception and reinforcement learning. *Neural Computation,* 2(4), 1990.