Image Processing in Cognitive Models with SegMan

Robert St. Amant

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292
stamant@csc.ncsu.edu

Mark O. Riedl

Institute for Creative Technologies
University of Southern California
Marina del Rey, CA 90292
riedl@ict.edu

Frank E. Ritter and Andrew Reifers

School of Information Sciences and Technology
Pennsylvania State University
University Park, PA 16802
frank.ritter@psu.edu

Abstract

A continuing trend in cognitive modeling research involves the application of theoretical results toward the improvement of human-computer interfaces. Our recent work has focused on the issue of making information about the often complex environments with which people interact directly available to cognitive models, so that the models can be used to evaluate interaction methods, interfaces, and theories of cognition. We have developed a system called SegMan (for Segmentation/Manipulation) that translates pixel-level input from a computer screen into the objects and symbols that a cognitive model can manipulate; SegMan further translates the manual output of a cognitive model, such as mouse movements and key presses, into a form that can be executed appropriately by a computer. The capabilities provided by SegMan have brought cognitive models directly into a number of interactive domains in which previous work has been forced to rely on simulations and abstract specifications. SegMan makes it possible for a cognitive model to interact with off-the-shelf applications, under some restrictions. Domains in which SegMan has been effective include productivity applications, computer games, desktop telephone interfaces, and robot navigation interfaces. This paper presents a survey of domains in which we have applied SegMan to support cognitive model-based evaluation.

1 Introduction

Over the past several years, researchers at North Carolina State University and the Pennsylvania State University have been developing cognitive models that can interact with increasingly realistic environments. This work has been based on a software substrate, called SegMan, that translates pixel-level input from a computer screen into the objects and symbols that a cognitive model can manipulate; SegMan further translates the manual output of a cognitive model, such as mouse movements and key presses, into a form that can be executed appropriately by a computer. While these capabilities exist in large part in other systems, SegMan is notable in that it minimizes the integration required of a cognitive model and its environment. The eventual goal of research on SegMan is to support the development of a cognitive-model-in-a-box for evaluating user interfaces, a black box that can simply be plugged into an existing interactive system, with a minimum of fuss, to evaluate it as a surrogate user.

In this paper, we give an overview of the history of this work. We present a number of off-the-shelf applications for which we have built cognitive models to control. The models are of varying complexity; our focus for much of this research was on how information could be passed between the cognitive model and the application. In the following section we explain how the image processing portion of SegMan works in some detail. Section 3 shows a progression of results in image processing for cognitive modelling in support of our general argument that cognitive models are ready for practical use in a wide variety of evaluation contexts.

2 Image processing in SegMan

The SegMan system is designed to be loosely connected with models based on cognitive architectures such as ACT-R (Anderson & Lebiere, 1998), Soar (Newell, 1990), or EPIC (Kieras & Meyer, 1997). A diagram of the integration is shown in Figure 1. Conceptually, SegMan provides a cognitive model (or indeed any programmatic controller) with a set of sensors and effectors that abstract away from the details of interaction with a specific task environment. Sensors take pixel-level information and translate it into an internal representation, which in turn can be queried to produce symbolic information appropriate for an external controller. SegMan takes advantage of the features of the graphics used in most computer interfaces (St. Amant & Reidl, 2001). Effectors take commands issued by an

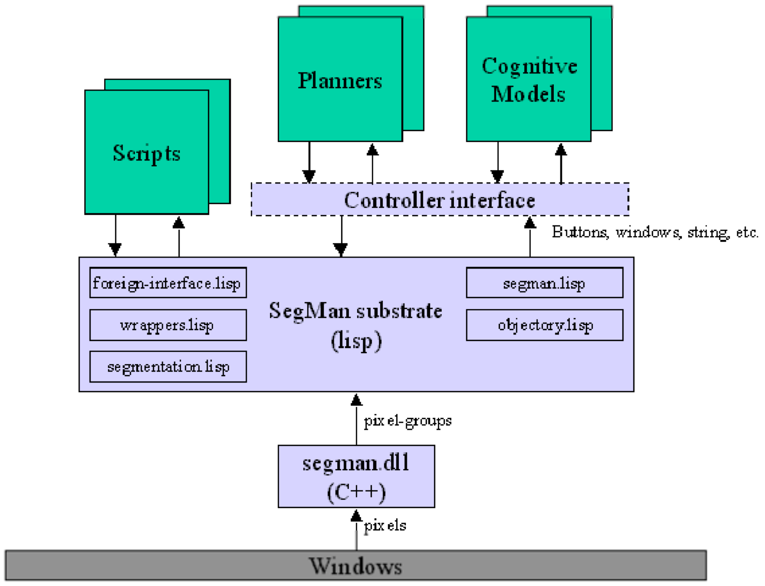


Figure 1. SegMan integration with a cognitive model.

external controller to its motors or motor equivalents, and translates them into operating system events. SegMan is currently limited to operation in user interfaces for Microsoft Windows 95/98/2000/XP.

Internally, SegMan uses simple computational vision routines to pick out features of interest in the Microsoft Windows graphical interface. The basic architecture of the SegMan system has a dynamic-link library (DLL), which is able to capture the screen as a bitmap and then process the bitmap into lists of pixel groups. A pixel group is a region of the screen where all pixels are like-colored. All pixels on the screen are assigned to non-overlapping pixel groups.

In Figure 2, in the diagram on the left, Group 1 is a pixel group composed of the pixels in the letter 'F'. Group 2 is the pixel group composed of the pixels in the dot. Group 3 is the pixel group composed of the pixels in the stem of the 'i'. Group 4 is the pixel group consisting of the background pixels. Pixel groups can be examined for specific shapes and for relationships between shapes. For shapes that consist of a single pixel group, such as the letter 'F', recognition is simple. One could either look at the arrangement of pixels within the group, or one could look at the pixel *neighbour numbers*.

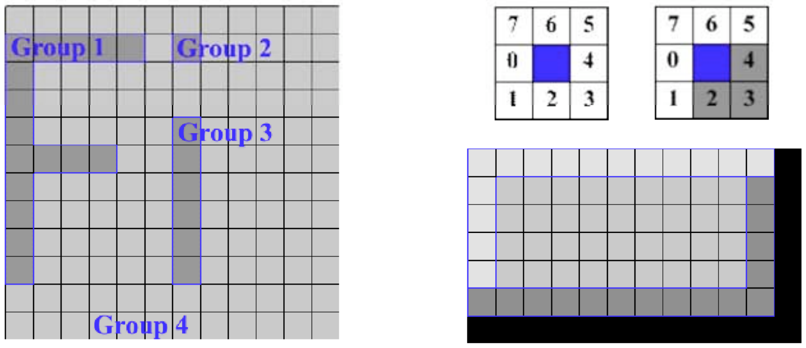


Figure 2. Pixel patterns

Each pixel group has an array of pixel neighbour numbers associated with it. The pixel neighbour numbers are encodings of the relationships between pixels within the group. Each pixel in a group has 0-8 neighbours, as shown on the upper right in Figure 2. Looking at an individual pixel in a pixel group, there are eight possible positions that a neighbour can be in: west, southwest, south, southeast, east, northeast, north, and northwest. We assign a numerical value to each neighbour position, respectively. Thus the west position is assigned to "0" and the northwest position is assigned to "7". Each of these numbers can be mapped to a bit in a single integer. Thus if a pixel is the top-right corner of a box, it has neighbours to the south (position 2), southeast (position 3), and to the east (position 4). The pixel neighbour value of that top-right corner pixel is $2^2 + 2^3 + 2^4 = 28$. To characterize an entire pixel group, based on the pixel neighbour integer associated with each pixel, we maintain a vector indexed by 0..255, each entry of which is the number of pixels in a pixel group with that specific pixel neighbour value. For example, in a pixel group that represents a 5 x 5 solid box, the inner 9 elements have a pixel neighbour value of 255; the entry at index 255 in the vector representing the pixel group is thus 9.

A pixel group can also be characterized by properties other than pixel neighbour properties, including the following:

- *count* is the number of pixels in the group;
- *size* is the area of the group's bounding box;
- *area* is the ratio of count to size;
- *height* is the height of the group's bounding box;
- *width* is the width of the group's bounding box;
- *red* is a component of the group's RGB value;
- *green* is a component of the group's RGB value;
- *blue* is a component of the group's RGB value;
- *color* is the group's numerical RGB value;
- *proportion* is the group's height divided by its width, or 0 if width is 0.

We define pixel patterns to capture combinations of group properties. A simple declarative form allows the definition of specific patterns. For example, a simple definition corresponding to a small box of a specific color and proportions can be expressed as given below, in which the unlabeled numbers correspond to pixel neighbor values. The definition can be entered in text form or automatically generated by a screen-parsing program, with specific patterns selected and named by the user.

```
(define-pattern small-box ()
 (:and  (:count 42) (:area 1) (:size 42) (:height 6) (:width 7)
   (:red 212) (:green 208) (:blue 200)
   (:proportion 5/6)
   7 28 (31 5) 112 (124 4) 193 (199 4) (241 5) (255 20)))
```

Segmentation of the screen into pixel groups gives us power in terms of recognizing features. However, simple segmentation only allows us to see shapes that consist of a single pixel group. Often it is valuable to recognize features that are made up of more than one pixel group. Examples of features made up of multiple pixel groups are icons, buttons, window borders, and strings of letters.

To recognize features that are not made up of a single pixel group we must employ a two-step process. The first step is to find the pixel groups that make up the feature. We do this by looking for specific pixel groups that might be part of overall feature. We do this by selecting pixel groups that have the right shape (the correct pixel neighbour numbers). Not all pixel groups with the correct shape are necessarily going to be part of the feature we are trying to detect. The second step is to choose from the candidate pixel groups the ones that are in proximity to each other and in the correct spatial configuration. The SegMan system provides a variety of functions that find pixel groups based on the spatial relationship to others.

For example, a standard Windows button is a rectilinear feature that appears to be raised out of the screen. This raised effect is created by applying a thin strip of color around the edges; lighter on the top and darker on the bottom. A sample is shown on the bottom right in Figure 2. As far as SegMan is concerned, a button is made up of three pixel groups: a rectangle and two L-shaped regions. However, these three groups must be in the correct relationship to each other in order to form what looks like a button. The lighter L-shape (upper shading) must be directly above and to the left of the rectangle and the darker L-shape (lower shading) must be directly below and to the right of the rectangle. When these relationships hold, there is a feature recognizable to the human use as a button.

In the first stage, we find all the pixel groups of the shapes we need: rectangles, upper-shading, and lower-shading. *Buttons* is an empty list into which we will collect all features that look like buttons. In the first stage, we iterate through the rectangles, looking for those in the proper relationship to the other shapes we have indicated. We find a pixel group in the *upper-shadings* list that most closely contains the rectangle. We find a pixel group in the *lower-shadings* list that most closely contains the rectangle. Containment is a useful relationship because, even though upper-shadings and lower-shadings are L-shaped, their bounding boxes enclose a much larger area that, ideally, will contain a rectangle if the feature is a button. The next check is proximity of the L-shapes to the rectangle. This is important because a button might be contained in a window and windows also are bounded by L-shaped shaded areas. But if the shading belonged to a window, one or both shadings will probably be further than five pixels away. Finally, we must make sure that the L-shape above the rectangle is lighter in color than the L-shape below the rectangle. If the upper L-shape was darker than the lower L-shape, perceptually, the feature will look recessed into the screen instead of raised.

3 Cognitive modelling domains for image processing

The image processing laid out in Section 2 is very simple in comparison with algorithms used for machine vision in demanding, real-world environments. As we shall see, however, SegMan's range of capabilities is surprisingly robust when applied to user interfaces, and even somewhat beyond.

3.1 Moving toward interaction with off-the-shelf software

Our earliest work on image processing of computer screens did not focus explicitly on cognitive modelling. Rather, we speculated that if an agent could observe and interpret all of the information seen by the user in interacting with a software application, the agent could provide better assistance, by providing advice or even offering to carry out tasks for the user. Our first demonstration of image processing capabilities for this purpose was in a system that could play the game of Windows Solitaire autonomously (Zettlemoyer & St. Amant, 1999). Solitaire was a reasonable target for our work: it has no application programming interface (API), which means that there is no other way for an agent to gain information about the state of the system except through the images presented on the screen; the internals of the system are not accessible to be re-written to better suit the needs of an external programmatic controller; Solitaire is possibly one of the most widely installed pieces of software in the world.

Figure 3 shows the interface to Solitaire. We developed a specification of the knowledge necessary to play Solitaire at a novice level, and implemented it in a language appropriate for an autonomous planning system to execute (St. Amant & Zettlemoyer, 2000). The system proved capable of recognizing most Windows icons, as well as the more specialized patterns representing card suits and ranks in Solitaire.

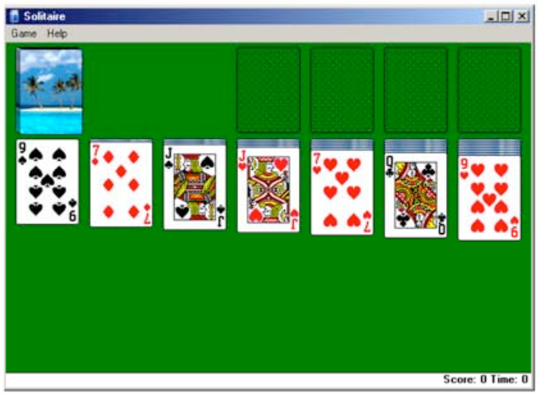
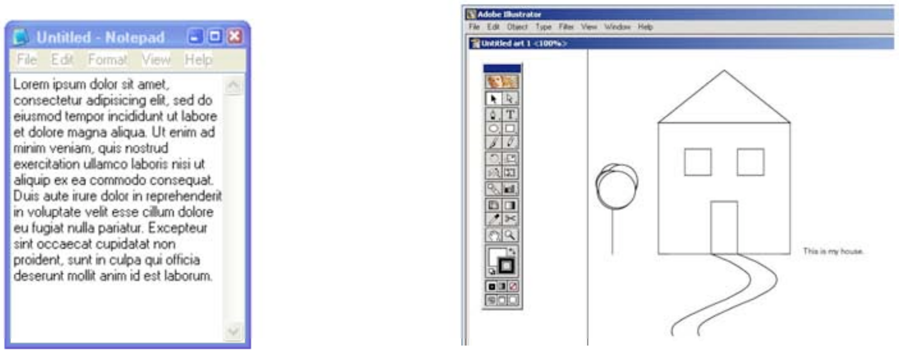


Figure 3. Solitaire user interface



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Figure 4. SegMan applications: Notepad and Illustrator

3.2 Interacting with productivity applications

It soon became clear, unfortunately, that the performance of our image processing algorithms was insufficient to support an interactive agent to assist users in the user interface. Nevertheless the autonomous behaviour of a program in interacting with off-the-shelf software proved appealing to cognitive modellers. We developed follow-on applications for SegMan, in the sense that we added new patterns to its library, enabling a controller to recognize a greater variety of interface widgets, and we extended the system's ability to perform character recognition of a few specific but common fonts. Figure 4 shows the next two applications we addressed: Microsoft Windows Notepad and Adobe Illustrator. Using SegMan, a simple programmatic controller in which squares, lines, circles, and other shapes could be represented produced the picture shown on the right. From a cognitive modelling standpoint, the interface on the left, Notepad, is more significant: we were able to reproduce the behaviour of a tutorial model based on the ACT-R architecture, using an external text editor as an experiment testbed, rather than a purpose-built interface.

3.3 Interacting with more dynamic environments

SegMan takes advantage of a number of properties of the user interface, in order to be effective. As visual environments, many user interfaces rely on relatively simple shapes that are composed of discrete visual elements that fall into predictable, static arrangements. For many software applications, however, such properties do not hold. We examined two new applications in order to test SegMan's effectiveness in dealing with more dynamic environments. The application shown on the left in Figure 5 is a Mars Rover game. The rover is shown by the icon in the middle left of the screen. It moves about its environment, avoiding obstacles such as the large rock above it and collecting samples from the beneath the smaller rocks to its right. Controlling the rover is straightforward for a human user: it involves selecting a direction via arrow keys and pressing an accelerator key to move the rover forward. This application was the first attempt to have SegMan engage in continuous control of the environment,

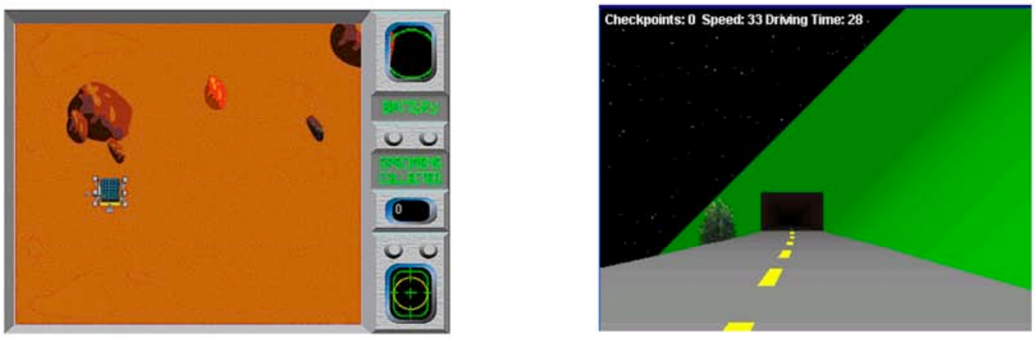


Figure 5. SegMan applications: roving on Mars and driving

with monitoring of the direction and location of the rover. From an image processing standpoint, the task is still simple, involving recognition of the front bumper of the rover and rocks that stand out from the background. Processing is complicated by the scrolling behaviour of the application when the rover reaches the edge of the display; this requires distinguishing objects seen for the first time from objects that have simply changed position due to a different viewpoint.

A driving game, shown on the right in Figure 5, was more challenging with respect to dynamic aspects of the environment. In this game, an accelerator, a brake, and steering controls are provided to the user, who must drive the simulated car around curves and through tunnels, avoiding cars in both lanes. A cognitive model we built for this purpose follows relatively simple heuristics for accelerating to an appropriate speed and staying in the centre of the lane. Image processing was again straightforward, though because the environment is constantly changing as the car moves forward, there were stronger performance constraints on SegMan (Shah et al., 2003). Our selection of specific visual features for SegMan to process was guided by observations of human driving behaviour (Land & Horwood, 1995; Land & Lee, 1994). These indicate that human drivers focus on specific areas of their field of view in deciding whether and when to turn. As with the rover game, it turned out that only a small number of visual properties are relevant for competence in this task. The edges of the road and the centre stripe provide sufficient information for the cognitive model to stay in its lane. Identification of this information is not trivial, due to changes in the colours of the surfaces in the environment as the car passes through tunnels, over bridges, and by roadside objects that partially obscure its view, but it turned out to be possible with the simple image processing performed by SegMan.

3.4 Interacting with more complex environments

To further test SegMan's ability to interact with off-the-shelf software we turned to a much more complex online visual interface, an online Casio's Roulette board, as shown in Figure 6. As with the interfaces in the previous section, the roulette interface requires real-time response, which imposes constraints on SegMan's processing time. The cognitive model used to drive this application is a relatively simple ACT-R model that collects visual information via specialized patterns devised for the roulette application. The model fires decision-making productions and uses SegMan's motor functionality to interact with the application. The model was based on a simple theory of gambling. The payout function is simple doubling-up scheme, which is successful only with infinite funds but is a losing strategy in reality. We found the successful implementation of an ACT-R model using SegMan in order to interact with a live online interface to be very encouraging. However, throughout the course of developing the model and integrating it with SegMan, it became apparent that SegMan is not yet capable of processing this complex of an interface in its entirety. The possible lessons and implications of this will be covered in Section 4.

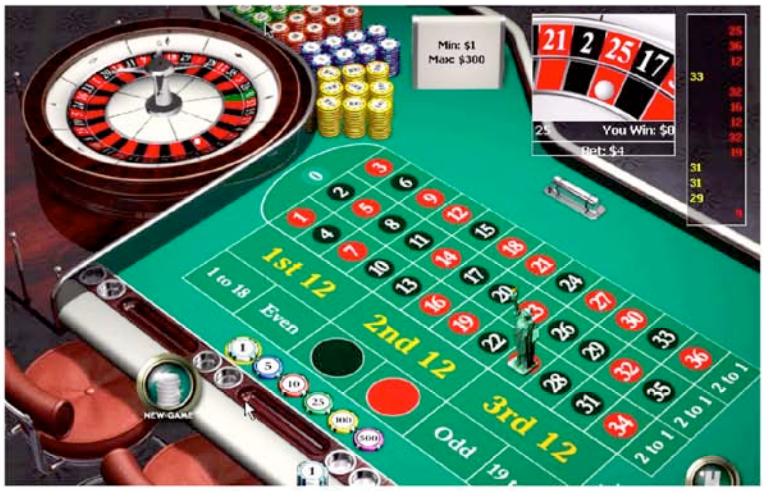


Figure 6. Roulette user interface

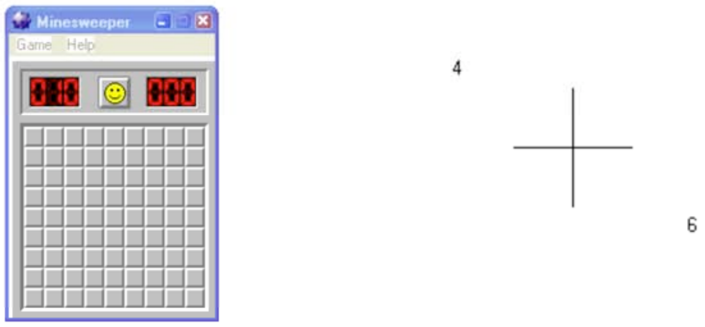


Figure 7. Soar and EPIC applications

3.5 Generalizing over cognitive modelling architectures

Up until this point, while we had been able to show that SegMan could provide control over a user interface to a cognitive model, a planning system, or an arbitrary program, our cognitive modelling work had been limited to the ACT-R architecture. We believed that it would be useful to demonstrate explicitly that SegMan could work with other modelling architectures, in particular Soar and EPIC. To do this we designed a simple Soar model to play Microsoft Minesweeper through the interface shown on the left in Figure 7. The model is able to play a game to completion, though it picks squares randomly. On the left in Figure 7 is the display of an image used in one of the cognitive models released with the EPIC system. The numbers and the crosshair provide targets for the model's focus of attention, which drives different actions to be taken. The model was run in a port of EPIC to Windows.

3.6 Modelling cell phone dialling

Consumer products are commonly being built around computational platforms that continue to decrease in size and expense. The increase in power that this gives consumers is often offset by increases in complexity. Cognitive models can help us better understand the usability properties of novel computational devices. In recent work we have examined cell phone interfaces. There are about a billion cell phones in use today, with this number growing rapidly. The most common activities on cell phones is dialling numbers, but modern phones include tools for managing contact information, voice mail, and hardware settings, and often software for playing games, browsing the Web, and connecting to specialized information services. Ensuring that users can make access such capabilities easily is a difficult problem.

We have developed models for dialing and for selecting menu items on phones. The models for dialing were developed for software interfaces to telephone simulations, as shown on the left in Figure 8 (St. Amant, Freed, & Ritter, 2004). Some of the questions that a cognitive model might answer concerning such interfaces include how quickly numbers can be dialed, how quickly other common procedures such as saving a number in memory can be carried out, and how many errors occur during different activities. Evaluation of these interfaces was possible using SegMan to provide information about button spacing, labels, and so forth to a cognitive model. In other work, we focused on user performance on real cell phones. We applied SegMan to the problem by analyzing the geometrical properties of digital images of cell phones, as shown on the right in Figure 8 (St. Amant, Horton, & Ritter, 2004). Despite the fact that SegMan was originally designed for artificial environments, its image processing approach worked reasonably well. In an informal test based on 12 images downloaded from the Web, of cell phones from different manufacturers with different keypad styles, we used SegMan in an attempt to identify the 12 standard keys. For six images, all standard keys were identified correctly, while in two others, 10 of 12 keys were identified correctly (errors involved merging two neighbouring keys.) For four images, SegMan was unable to identify any keys—we believe that the failures were due to such factors as variation in background color, lighting artifacts, and low distinguishability of key borders.

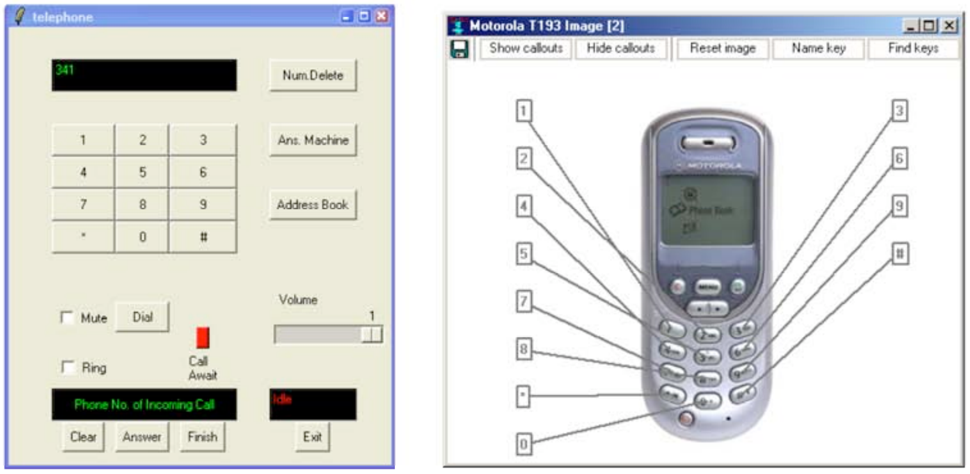
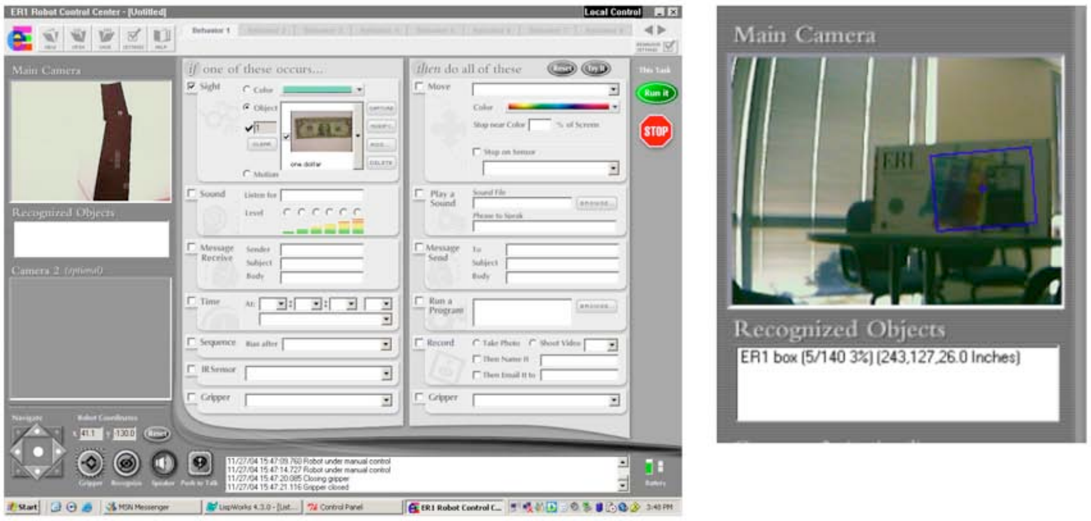


Figure 8. A desktop telephone interface and a cell phone image, after automatic identification of keys

3.7 Modelling interactive robot navigation

In our most recent work, we have built cognitive models for remote control of a simple mobile robot. Our robot is an ER1 Personal Robot System, a simple hobbyist robot marketed by Evolution Robotics™. While the ER1 is an entry-level robot lacking highly specialized technological capabilities, it has many features analogous to those in more expensive field robots and can perform comparable tasks. On the 24 x 16 x 15 inch ER1, a laptop is placed running Windows XP to perform the robot's onboard computing. The robot is teleoperated via another laptop, using a wireless internet connection. The ER1 robot Control Center software is installed on both laptops, which supports development and debugging. The laptop on the robot platform is connected via USB cables to the robot's camera, its gripper, and the motors that drive its wheels.

The software Control Center is shown in Figure 9, with a detail view shown on the right. On the upper left of its display is the camera view from the ball-shaped camera on the top of the robot. The navigation buttons for



```
11/27/04 15:47:09.760 Robot under manual control
11/27/04 15:47:14.727 Robot under manual control
11/27/04 15:47:20.085 Closing gripper
11/27/04 15:47:21.116 Gripper closed
```

ER1 box (5/140 3%) (243,127,26.0 Inches)

Figure 9. ER1 robot controller interface

controlling the robot's movement are on the bottom right. Users can drive the robot in three ways: (a) use the arrow keys on the keyboard; (b) click on the navigation buttons; and (c) use the software joystick that appears as a dot in the middle of the navigation buttons. The camera view in the left image in Figure 9 shows a navigation task for a cognitive model that uses SegMan: the model follows a marked path to a goal location, at which point a simple grasping task is accomplished by the robot. The black line is provided by tape on a white background, to eliminate such problems as lighting variation for SegMan's image processing. On the right of Figure 9 a different task is shown that illustrates the potential for SegMan's interaction with existing software. The ER1 includes its own pattern recognition capabilities, which here are applied to the identification of a pattern on its shipping box. Rather than superseding this functionality, SegMan simply identifies the blue rectangle surrounding the recognized object, and passes this information to a cognitive model controller. SegMan can also read the text provided by the ER1's control centre in the box below the camera view, in order to learn specifically what kind of object the robot has identified. SegMan has also been used to create two other models that program the ER1 interface to open and close the gripper when there is an object in it, and to have the robot speak "I see a dollar" when a dollar bill is recognized (Kukreja, 2004).

4 Discussion

The wide range of examples given in Section 3 point to a number of lessons we can learn from our attempts to make information from rich visual environments available to cognitive models.

- A more detailed and cognitively accurate account of visual processing is needed in SegMan. The image processing approach laid out in Section 2, while adequate for many purposes, does not necessarily extend to environments outside standard user interfaces. A significant step forward, for example, would involve generalization over recognized patterns, via machine learning, such that SegMan could recognize objects that were not already part of its library of templates. The internal representation of objects in terms of pixel neighbour counts and pixel group properties is not rich enough for such generalization. Creating these patterns by hand is possible for small tasks, but is difficult for large ones.
- Engineering appropriate task-dependent visual representations remains a barrier. While SegMan, even in its current form, provides a useful set of building blocks from which patterns for recognizing complex objects can be constructed, this is still a time-consuming and error-prone process. The difficulty is that these building blocks are at too low a level of abstraction.
- Complex interfaces cannot be handled with generality. One lesson we learned about SegMan was that it has difficulty processing complex interfaces. Although this is correctly viewed as a shortcoming of SegMan, it is also in accordance with perceptual theories (Bartels & Zeki, 1998). In short humans do not process interfaces in their entirety. Rather, there is a complex interaction of attention allocation and focalized attention that makes up human visual processing. Currently SegMan's algorithms process all areas of the screen with the same level of acuity. SegMan has progressed to a point where it could greatly benefit from both attention allocation and foveal increased acuity. Not only would this greatly improve SegMan's ability to accurately test and critique interface design, but it would also be a large step for the development of an accurate unified theory of perception.

Despite its limitations, SegMan has proved to be a useful addition to cognitive modelling technology. It has led us into areas that have not yet been explored in cognitive modelling research, and we believe that the future of such integrative approaches to cognitive modelling is bright.

Acknowledgements

The projects reported here were partially supported by NSF (Foundation (IIS-0083281), the Office of Naval Research (N00014-02-1-0021, and by the Space and Naval Warfare Systems Center, San Diego.

References

Anderson, J., and Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Lawrence Erlbaum.

Bartels, A., and Zeki, S. (1998). *The theory of multistage integration in the visual brain.* Paper presented at the Proceedings of the Royal Society.

Kieras, D. E., and Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*.

Kukreja, U. (2004). *Towards model-based evaluations of human-robot interfaces*. Unpublished MS thesis, Department of Computer Science and Engineering, The Pennsylvania State University.

Land, M. F., and Horwood, J. (1995). Which parts of the road guide steering? *Nature, 377*:339-340.

Land, M. F., and Lee, D. N. (1994). Where we look when we steer. *Nature, 369*:742-744.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

Riedl, M. O., and St. Amant, R. (2002). Toward automated exploration of interactive systems. *Proceedings of the International Conference on Intelligent User Interfaces*. Pp. 135-142.

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7*(2), 141-173.

Ritter, F. E., Van Rooy, D., St. Amant, R., and Simpson, K. Providing user models with direct access to computer interfaces: An exploratory study of a simple human-robot interface. Under review.

Ritter, F. E., Van Rooy, D., & St. Amant, R. (2002). A user modeling design tool based on a cognitive architecture for comparing interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002*. 111-118. Kluwer Academics Publisher, Dordrecht.

Shah, K., Rajyaguru, S., St. Amant, R., and Ritter, F. E. (2003). Connecting a cognitive model to dynamic gaming environments: Architectural and image processing issues. *Proceedings of the Fifth International Conference on Cognitive Modeling (ICCM)*. Pp. 189-194.

St. Amant, R. (2000). Interface agents as surrogate users. *Intelligence magazine*. 11(2): 29-38. Summer.

St. Amant, R., Freed A., and Ritter, F. E. (2004). Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research 6*(1): 71-88.

St. Amant, R., Horton, T. E., & Ritter, F. E. (2004). Model-based evaluation of cell phone menu interaction. In *Proceedings of the CHI'04 Conference on Human Factors in Computer Systems*. 343-350. New York, NY: ACM.

St. Amant, R., Lieberman, H., Potter, R., and Zettlemoyer, L. S. (2000). Visual generalization in programming by example. *Communications of the ACM, 43*(3): 107-114. March.

St. Amant, R., and Dudani, A. (2002). An environment for programming user interface softbots. *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI)*. Pp. 119-122.

St. Amant, R., and Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies 55*(1): 15-39.

St. Amant, R., and Zettlemoyer, L. S. (2000). The user interface as an agent environment. *Proceedings of the International Conference on Autonomous Agents*. Pp. 483-490.

Zettlemoyer, L. S., and St. Amant, R. (1999). A visual medium for programmatic control of interactive applications. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. Pp. 199-206.

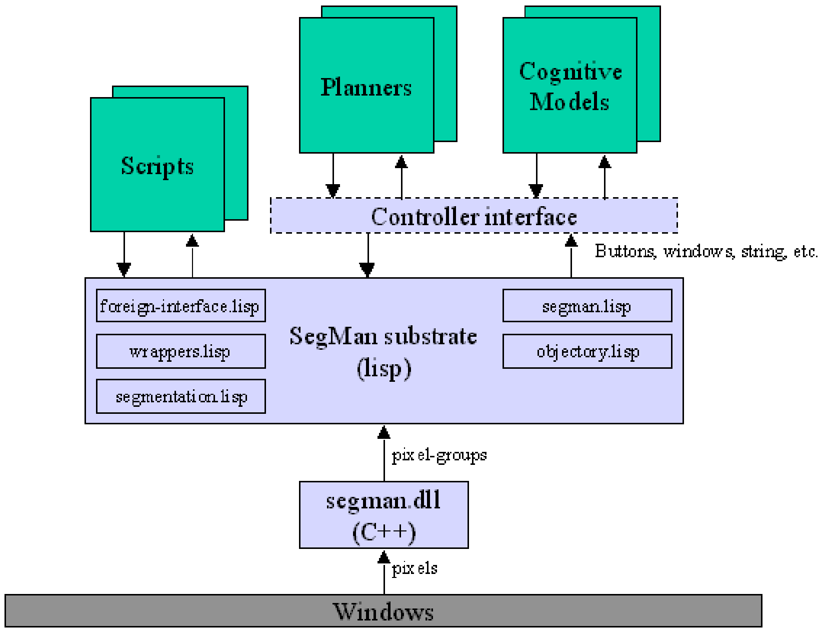


Figure 1. SegMan integration with a cognitive model.

external controller to its motors or motor equivalents, and translates them into operating system events. SegMan is currently limited to operation in user interfaces for Microsoft Windows 95/98/2000/XP.

Internally, SegMan uses simple computational vision routines to pick out features of interest in the Microsoft Windows graphical interface. The basic architecture of the SegMan system has a dynamic-link library (DLL), which is able to capture the screen as a bitmap and then process the bitmap into lists of pixel groups. A pixel group is a region of the screen where all pixels are like-colored. All pixels on the screen are assigned to non-overlapping pixel groups.

In Figure 2, in the diagram on the left, Group 1 is a pixel group composed of the pixels in the letter 'F'. Group 2 is the pixel group composed of the pixels in the dot. Group 3 is the pixel group composed of the pixels in the stem of the 'i'. Group 4 is the pixel group consisting of the background pixels. Pixel groups can be examined for specific shapes and for relationships between shapes. For shapes that consist of a single pixel group, such as the letter 'F', recognition is simple. One could either look at the arrangement of pixels within the group, or one could look at the pixel *neighbour numbers*.

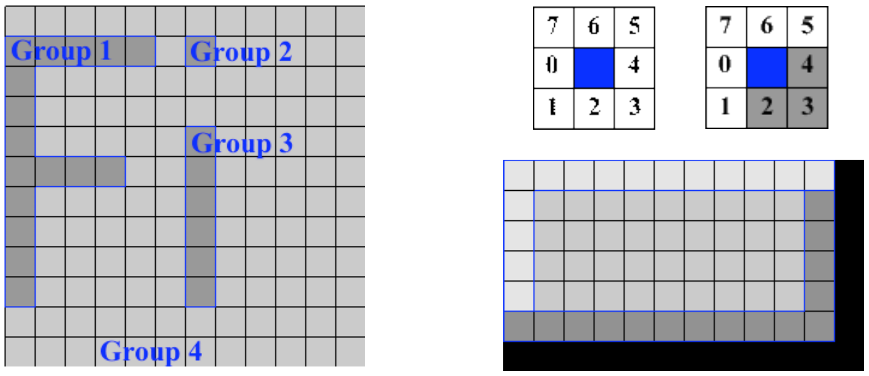


Figure 2. Pixel patterns

Each pixel group has an array of pixel neighbour numbers associated with it. The pixel neighbour numbers are encodings of the relationships between pixels within the group. Each pixel in a group has 0-8 neighbours, as shown on the upper right in Figure 2. Looking at an individual pixel in a pixel group, there are eight possible positions that a neighbour can be in: west, southwest, south, southeast, east, northeast, north, and northwest. We assign a numerical value to each neighbour position, respectively. Thus the west position is assigned to "0" and the northwest position is assigned to "7". Each of these numbers can be mapped to a bit in a single integer. Thus if a pixel is the top-right corner of a box, it has neighbours to the south (position 2), southeast (position 3), and to the east (position 4). The pixel neighbour value of that top-right corner pixel is $2^2 + 2^3 + 2^4 = 28$. To characterize an entire pixel group, based on the pixel neighbour integer associated with each pixel, we maintain a vector indexed by 0..255, each entry of which is the number of pixels in a pixel group with that specific pixel neighbour value. For example, in a pixel group that represents a 5 x 5 solid box, the inner 9 elements have a pixel neighbour value of 255; the entry at index 255 in the vector representing the pixel group is thus 9.

A pixel group can also be characterized by properties other than pixel neighbour properties, including the following:

- *count* is the number of pixels in the group;
- *size* is the area of the group's bounding box;
- *area* is the ratio of count to size;
- *height* is the height of the group's bounding box;
- *width* is the width of the group's bounding box;
- *red* is a component of the group's RGB value;
- *green* is a component of the group's RGB value;
- *blue* is a component of the group's RGB value;
- *color* is the group's numerical RGB value;
- *proportion* is the group's height divided by its width, or 0 if width is 0.

We define pixel patterns to capture combinations of group properties. A simple declarative form allows the definition of specific patterns. For example, a simple definition corresponding to a small box of a specific color and proportions can be expressed as given below, in which the unlabeled numbers correspond to pixel neighbor values. The definition can be entered in text form or automatically generated by a screen-parsing program, with specific patterns selected and named by the user.

```
(define-pattern small-box ()
  (:and   (:count 42) (:area 1) (:size 42) (:height 6) (:width 7)
     (:red 212) (:green 208) (:blue 200)
     (:proportion 5/6)
     7 28 (31 5) 112 (124 4) 193 (199 4) (241 5) (255 20)))
```

Segmentation of the screen into pixel groups gives us power in terms of recognizing features. However, simple segmentation only allows us to see shapes that consist of a single pixel group. Often it is valuable to recognize features that are made up of more than one pixel group. Examples of features made up of multiple pixel groups are icons, buttons, window borders, and strings of letters.

To recognize features that are not made up of a single pixel group we must employ a two-step process. The first step is to find the pixel groups that make up the feature. We do this by looking for specific pixel groups that might be part of overall feature. We do this by selecting pixel groups that have the right shape (the correct pixel neighbour numbers). Not all pixel groups with the correct shape are necessarily going to be part of the feature we are trying to detect. The second step is to choose from the candidate pixel groups the ones that are in proximity to each other and in the correct spatial configuration. The SegMan system provides a variety of functions that find pixel groups based on the spatial relationship to others.

For example, a standard Windows button is a rectilinear feature that appears to be raised out of the screen. This raised effect is created by applying a thin strip of color around the edges; lighter on the top and darker on the bottom. A sample is shown on the bottom right in Figure 2. As far as SegMan is concerned, a button is made up of three pixel groups: a rectangle and two L-shaped regions. However, these three groups must be in the correct relationship to each other in order to form what looks like a button. The lighter L-shape (upper shading) must be directly above and to the left of the rectangle and the darker L-shape (lower shading) must be directly below and to the right of the rectangle. When these relationships hold, there is a feature recognizable to the human use as a button.

In the first stage, we find all the pixel groups of the shapes we need: rectangles, upper-shading, and lower-shading. *Buttons* is an empty list into which we will collect all features that look like buttons. In the first stage, we iterate through the rectangles, looking for those in the proper relationship to the other shapes we have indicated. We find a pixel group in the *upper-shadings* list that most closely contains the rectangle. We find a pixel group in the *lower-shadings* list that most closely contains the rectangle. Containment is a useful relationship because, even though upper-shadings and lower-shadings are L-shaped, their bounding boxes enclose a much larger area that, ideally, will contain a rectangle if the feature is a button. The next check is proximity of the L-shapes to the rectangle. This is important because a button might be contained in a window and windows also are bounded by L-shaped shaded areas. But if the shading belonged to a window, one or both shadings will probably be further than five pixels away. Finally, we must make sure that the L-shape above the rectangle is lighter in color than the L-shape below the rectangle. If the upper L-shape was darker than the lower L-shape, perceptually, the feature will look recessed into the screen instead of raised.

3 Cognitive modelling domains for image processing

The image processing laid out in Section 2 is very simple in comparison with algorithms used for machine vision in demanding, real-world environments. As we shall see, however, SegMan's range of capabilities is surprisingly robust when applied to user interfaces, and even somewhat beyond.

3.1 Moving toward interaction with off-the-shelf software

Our earliest work on image processing of computer screens did not focus explicitly on cognitive modelling. Rather, we speculated that if an agent could observe and interpret all of the information seen by the user in interacting with a software application, the agent could provide better assistance, by providing advice or even offering to carry out tasks for the user. Our first demonstration of image processing capabilities for this purpose was in a system that could play the game of Windows Solitaire autonomously (Zettlemoyer & St. Amant, 1999). Solitaire was a reasonable target for our work: it has no application programming interface (API), which means that there is no other way for an agent to gain information about the state of the system except through the images presented on the screen; the internals of the system are not accessible to be re-written to better suit the needs of an external programmatic controller; Solitaire is possibly one of the most widely installed pieces of software in the world.

Figure 3 shows the interface to Solitaire. We developed a specification of the knowledge necessary to play Solitaire at a novice level, and implemented it in a language appropriate for an autonomous planning system to execute (St. Amant & Zettlemoyer, 2000). The system proved capable of recognizing most Windows icons, as well as the more specialized patterns representing card suits and ranks in Solitaire.

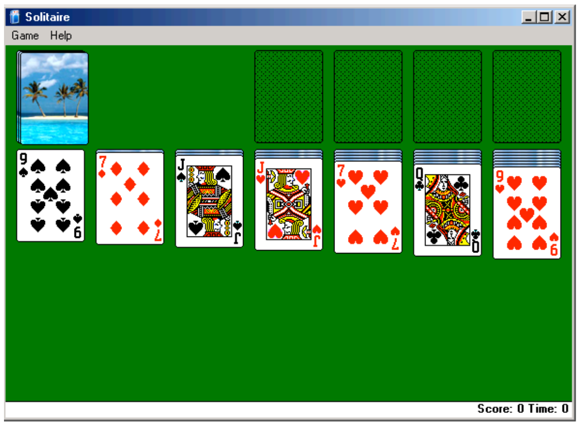
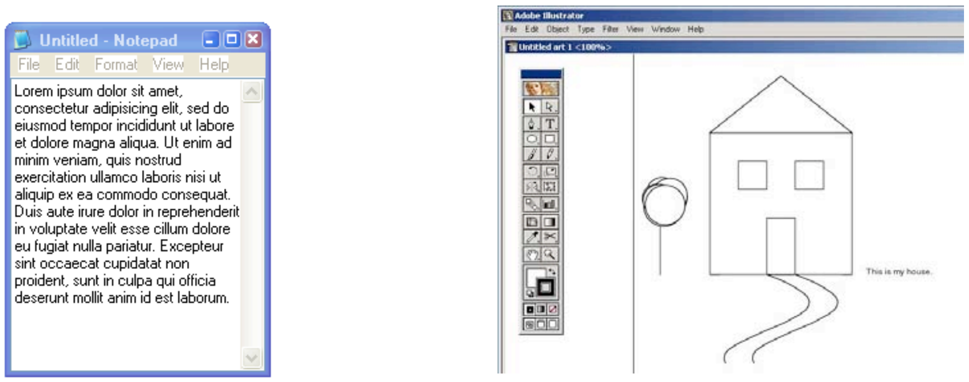


Figure 3. Solitaire user interface



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

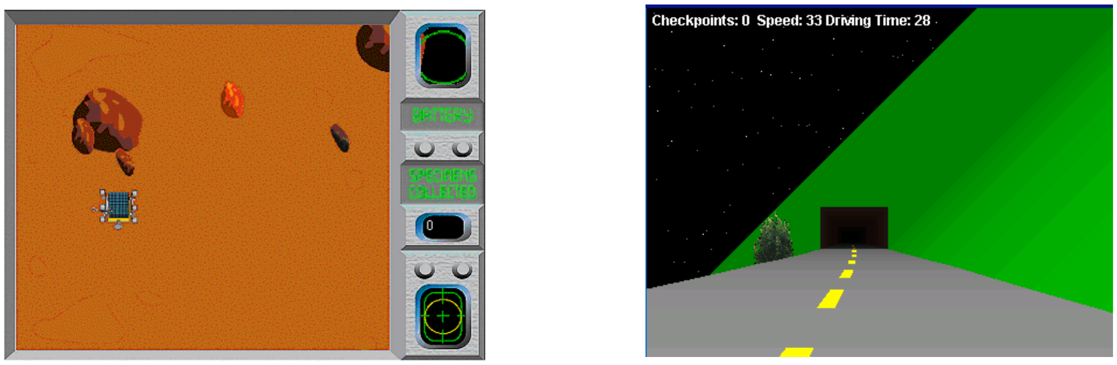
Figure 4. SegMan applications: Notepad and Illustrator

3.2 Interacting with productivity applications

It soon became clear, unfortunately, that the performance of our image processing algorithms was insufficient to support an interactive agent to assist users in the user interface. Nevertheless the autonomous behaviour of a program in interacting with off-the-shelf software proved appealing to cognitive modellers. We developed follow-on applications for SegMan, in the sense that we added new patterns to its library, enabling a controller to recognize a greater variety of interface widgets, and we extended the system's ability to perform character recognition of a few specific but common fonts. Figure 4 shows the next two applications we addressed: Microsoft Windows Notepad and Adobe Illustrator. Using SegMan, a simple programmatic controller in which squares, lines, circles, and other shapes could be represented produced the picture shown on the right. From a cognitive modelling standpoint, the interface on the left, Notepad, is more significant: we were able to reproduce the behaviour of a tutorial model based on the ACT-R architecture, using an external text editor as an experiment testbed, rather than a purpose-built interface.

3.3 Interacting with more dynamic environments

SegMan takes advantage of a number of properties of the user interface, in order to be effective. As visual environments, many user interfaces rely on relatively simple shapes that are composed of discrete visual elements that fall into predictable, static arrangements. For many software applications, however, such properties do not hold. We examined two new applications in order to test SegMan's effectiveness in dealing with more dynamic environments. The application shown on the left in Figure 5 is a Mars Rover game. The rover is shown by the icon in the middle left of the screen. It moves about its environment, avoiding obstacles such as the large rock above it and collecting samples from the beneath the smaller rocks to its right. Controlling the rover is straightforward for a human user: it involves selecting a direction via arrow keys and pressing an accelerator key to move the rover forward. This application was the first attempt to have SegMan engage in continuous control of the environment,



Checkpoints: 0 Speed: 33 Driving Time: 28

Figure 5. SegMan applications: roving on Mars and driving

with monitoring of the direction and location of the rover. From an image processing standpoint, the task is still simple, involving recognition of the front bumper of the rover and rocks that stand out from the background. Processing is complicated by the scrolling behaviour of the application when the rover reaches the edge of the display; this requires distinguishing objects seen for the first time from objects that have simply changed position due to a different viewpoint.

A driving game, shown on the right in Figure 5, was more challenging with respect to dynamic aspects of the environment. In this game, an accelerator, a brake, and steering controls are provided to the user, who must drive the simulated car around curves and through tunnels, avoiding cars in both lanes. A cognitive model we built for this purpose follows relatively simple heuristics for accelerating to an appropriate speed and staying in the centre of the lane. Image processing was again straightforward, though because the environment is constantly changing as the car moves forward, there were stronger performance constraints on SegMan (Shah et al., 2003). Our selection of specific visual features for SegMan to process was guided by observations of human driving behaviour (Land & Horwood, 1995; Land & Lee, 1994). These indicate that human drivers focus on specific areas of their field of view in deciding whether and when to turn. As with the rover game, it turned out that only a small number of visual properties are relevant for competence in this task. The edges of the road and the centre stripe provide sufficient information for the cognitive model to stay in its lane. Identification of this information is not trivial, due to changes in the colours of the surfaces in the environment as the car passes through tunnels, over bridges, and by roadside objects that partially obscure its view, but it turned out to be possible with the simple image processing performed by SegMan.

3.4 Interacting with more complex environments

To further test SegMan's ability to interact with off-the-shelf software we turned to a much more complex online visual interface, an online Casio's Roulette board, as shown in Figure 6. As with the interfaces in the previous section, the roulette interface requires real-time response, which imposes constraints on SegMan's processing time. The cognitive model used to drive this application is a relatively simple ACT-R model that collects visual information via specialized patterns devised for the roulette application. The model fires decision-making productions and uses SegMan's motor functionality to interact with the application. The model was based on a simple theory of gambling. The payout function is simple doubling-up scheme, which is successful only with infinite funds but is a losing strategy in reality. We found the successful implementation of an ACT-R model using SegMan in order to interact with a live online interface to be very encouraging. However, throughout the course of developing the model and integrating it with SegMan, it became apparent that SegMan is not yet capable of processing this complex of an interface in its entirety. The possible lessons and implications of this will be covered in Section 4.

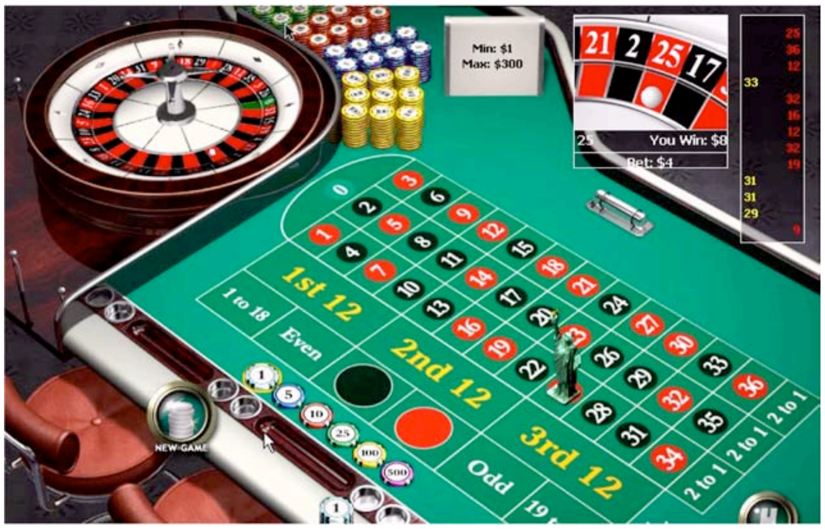
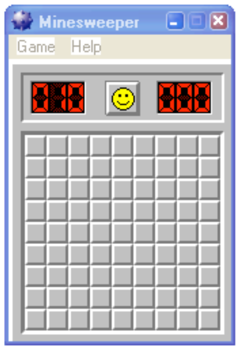


Figure 6. Roulette user interface



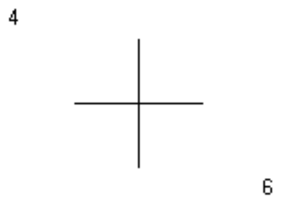


Figure 7. Soar and EPIC applications

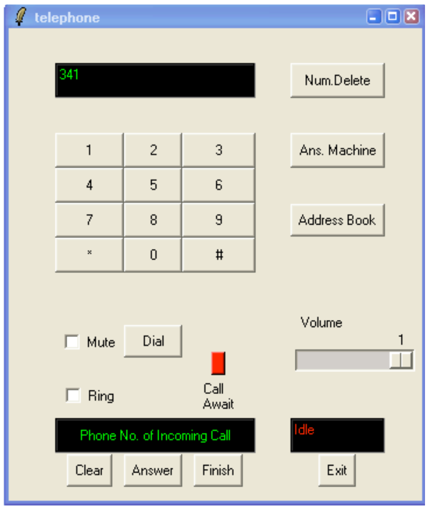
3.5 Generalizing over cognitive modelling architectures

Up until this point, while we had been able to show that SegMan could provide control over a user interface to a cognitive model, a planning system, or an arbitrary program, our cognitive modelling work had been limited to the ACT-R architecture. We believed that it would be useful to demonstrate explicitly that SegMan could work with other modelling architectures, in particular Soar and EPIC. To do this we designed a simple Soar model to play Microsoft Minesweeper through the interface shown on the left in Figure 7. The model is able to play a game to completion, though it picks squares randomly. On the left in Figure 7 is the display of an image used in one of the cognitive models released with the EPIC system. The numbers and the crosshair provide targets for the model's focus of attention, which drives different actions to be taken. The model was run in a port of EPIC to Windows.

3.6 Modelling cell phone dialling

Consumer products are commonly being built around computational platforms that continue to decrease in size and expense. The increase in power that this gives consumers is often offset by increases in complexity. Cognitive models can help us better understand the usability properties of novel computational devices. In recent work we have examined cell phone interfaces. There are about a billion cell phones in use today, with this number growing rapidly. The most common activities on cell phones is dialling numbers, but modern phones include tools for managing contact information, voice mail, and hardware settings, and often software for playing games, browsing the Web, and connecting to specialized information services. Ensuring that users can make access such capabilities easily is a difficult problem.

We have developed models for dialing and for selecting menu items on phones. The models for dialing were developed for software interfaces to telephone simulations, as shown on the left in Figure 8 (St. Amant, Freed, & Ritter, 2004). Some of the questions that a cognitive model might answer concerning such interfaces include how quickly numbers can be dialed, how quickly other common procedures such as saving a number in memory can be carried out, and how many errors occur during different activities. Evaluation of these interfaces was possible using SegMan to provide information about button spacing, labels, and so forth to a cognitive model. In other work, we focused on user performance on real cell phones. We applied SegMan to the problem by analyzing the geometrical properties of digital images of cell phones, as shown on the right in Figure 8 (St. Amant, Horton, & Ritter, 2004). Despite the fact that SegMan was originally designed for artificial environments, its image processing approach worked reasonably well. In an informal test based on 12 images downloaded from the Web, of cell phones from different manufacturers with different keypad styles, we used SegMan in an attempt to identify the 12 standard keys. For six images, all standard keys were identified correctly, while in two others, 10 of 12 keys were identified correctly (errors involved merging two neighbouring keys.) For four images, SegMan was unable to identify any keys—we believe that the failures were due to such factors as variation in background color, lighting artifacts, and low distinguishability of key borders.



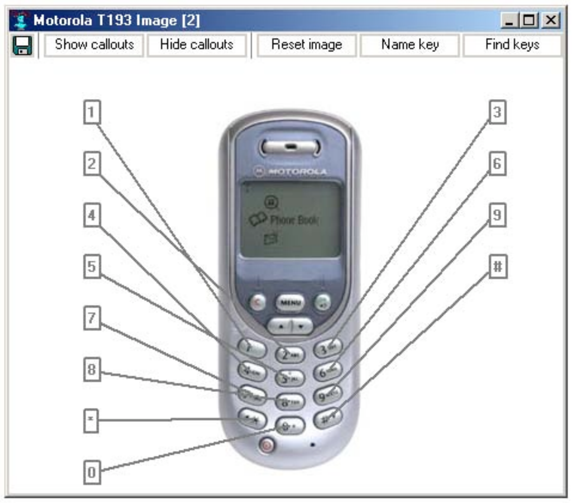
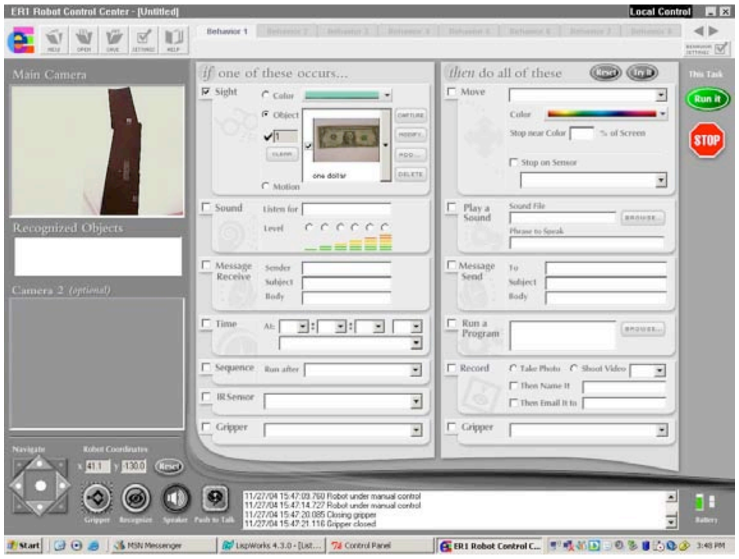


Figure 8. A desktop telephone interface and a cell phone image, after automatic identification of keys

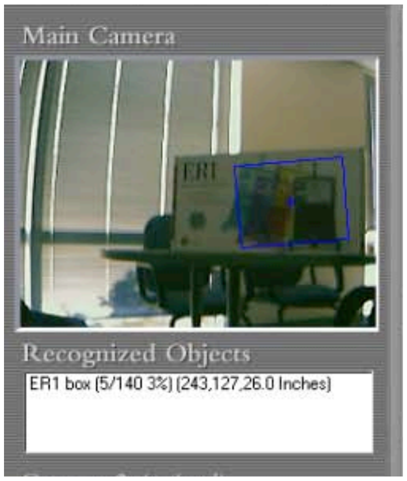
3.7 Modelling interactive robot navigation

In our most recent work, we have built cognitive models for remote control of a simple mobile robot. Our robot is an ER1 Personal Robot System, a simple hobbyist robot marketed by Evolution Robotics™. While the ER1 is an entry-level robot lacking highly specialized technological capabilities, it has many features analogous to those in more expensive field robots and can perform comparable tasks. On the 24 x 16 x 15 inch ER1, a laptop is placed running Windows XP to perform the robot's onboard computing. The robot is teleoperated via another laptop, using a wireless internet connection. The ER1 robot Control Center software is installed on both laptops, which supports development and debugging. The laptop on the robot platform is connected via USB cables to the robot's camera, its gripper, and the motors that drive its wheels.

The software Control Center is shown in Figure 9, with a detail view shown on the right. On the upper left of its display is the camera view from the ball-shaped camera on the top of the robot. The navigation buttons for



```
11/27/04 15:47:09.760 Robot under manual control
11/27/04 15:47:14.727 Robot under manual control
11/27/04 15:47:20.085 Closing gripper
11/27/04 15:47:21.116 Gripper closed
```



ER1 box (5/140 3%) (243,127,26.0 Inches)

Figure 9. ER1 robot controller interface

controlling the robot's movement are on the bottom right. Users can drive the robot in three ways: (a) use the arrow keys on the keyboard; (b) click on the navigation buttons; and (c) use the software joystick that appears as a dot in the middle of the navigation buttons. The camera view in the left image in Figure 9 shows a navigation task for a cognitive model that uses SegMan: the model follows a marked path to a goal location, at which point a simple grasping task is accomplished by the robot. The black line is provided by tape on a white background, to eliminate such problems as lighting variation for SegMan's image processing. On the right of Figure 9 a different task is shown that illustrates the potential for SegMan's interaction with existing software. The ER1 includes its own pattern recognition capabilities, which here are applied to the identification of a pattern on its shipping box. Rather than superseding this functionality, SegMan simply identifies the blue rectangle surrounding the recognized object, and passes this information to a cognitive model controller. SegMan can also read the text provided by the ER1's control centre in the box below the camera view, in order to learn specifically what kind of object the robot has identified. SegMan has also been used to create two other models that program the ER1 interface to open and close the gripper when there is an object in it, and to have the robot speak "I see a dollar" when a dollar bill is recognized (Kukreja, 2004).

4 Discussion

The wide range of examples given in Section 3 point to a number of lessons we can learn from our attempts to make information from rich visual environments available to cognitive models.

- A more detailed and cognitively accurate account of visual processing is needed in SegMan. The image processing approach laid out in Section 2, while adequate for many purposes, does not necessarily extend to environments outside standard user interfaces. A significant step forward, for example, would involve generalization over recognized patterns, via machine learning, such that SegMan could recognize objects that were not already part of its library of templates. The internal representation of objects in terms of pixel neighbour counts and pixel group properties is not rich enough for such generalization. Creating these patterns by hand is possible for small tasks, but is difficult for large ones.
- Engineering appropriate task-dependent visual representations remains a barrier. While SegMan, even in its current form, provides a useful set of building blocks from which patterns for recognizing complex objects can be constructed, this is still a time-consuming and error-prone process. The difficulty is that these building blocks are at too low a level of abstraction.
- Complex interfaces cannot be handled with generality. One lesson we learned about SegMan was that it has difficulty processing complex interfaces. Although this is correctly viewed as a shortcoming of SegMan, it is also in accordance with perceptual theories (Bartels & Zeki, 1998). In short humans do not process interfaces in their entirety. Rather, there is a complex interaction of attention allocation and focalized attention that makes up human visual processing. Currently SegMan's algorithms process all areas of the screen with the same level of acuity. SegMan has progressed to a point where it could greatly benefit from both attention allocation and foveal increased acuity. Not only would this greatly improve SegMan's ability to accurately test and critique interface design, but it would also be a large step for the development of an accurate unified theory of perception.

Despite its limitations, SegMan has proved to be a useful addition to cognitive modelling technology. It has led us into areas that have not yet been explored in cognitive modelling research, and we believe that the future of such integrative approaches to cognitive modelling is bright.

Acknowledgements

The projects reported here were partially supported by NSF (Foundation (IIS-0083281), the Office of Naval Research (N00014-02-1-0021, and by the Space and Naval Warfare Systems Center, San Diego.

References

Anderson, J., and Lebiere, C. (1998). *The atomic components of thought.* Mahwah, NJ: Lawrence Erlbaum.

Bartels, A., and Zeki, S. (1998). *The theory of multistage integration in the visual brain.* Paper presented at the Proceedings of the Royal Society.

Kieras, D. E., and Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-Computer Interaction*.

Kukreja, U. (2004). *Towards model-based evaluations of human-robot interfaces*. Unpublished MS thesis, Department of Computer Science and Engineering, The Pennsylvania State University.

Land, M. F., and Horwood, J. (1995). Which parts of the road guide steering? *Nature, 377*:339-340.

Land, M. F., and Lee, D. N. (1994). Where we look when we steer. *Nature, 369*:742-744.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

Riedl, M. O., and St. Amant, R. (2002). Toward automated exploration of interactive systems. *Proceedings of the International Conference on Intelligent User Interfaces*. Pp. 135-142.

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7*(2), 141-173.

Ritter, F. E., Van Rooy, D., St. Amant, R., and Simpson, K. Providing user models with direct access to computer interfaces: An exploratory study of a simple human-robot interface. Under review.

Ritter, F. E., Van Rooy, D., & St. Amant, R. (2002). A user modeling design tool based on a cognitive architecture for comparing interfaces. In C. Kolski & J. Vanderdonckt (Eds.), *Computer-Aided Design of User Interfaces III, Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces CADUI'2002*. 111-118. Kluwer Academics Publisher, Dordrecht.

Shah, K., Rajyaguru, S., St. Amant, R., and Ritter, F. E. (2003). Connecting a cognitive model to dynamic gaming environments: Architectural and image processing issues. *Proceedings of the Fifth International Conference on Cognitive Modeling (ICCM)*. Pp. 189-194.

St. Amant, R. (2000). Interface agents as surrogate users. *Intelligence magazine.* 11(2): 29-38. Summer.

St. Amant, R., Freed A., and Ritter, F. E. (2004). Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research 6*(1): 71-88.

St. Amant, R., Horton, T. E., & Ritter, F. E. (2004). Model-based evaluation of cell phone menu interaction. In *Proceedings of the CHI'04 Conference on Human Factors in Computer Systems*. 343-350. New York, NY: ACM.

St. Amant, R., Lieberman, H., Potter, R., and Zettlemoyer, L. S. (2000). Visual generalization in programming by example. *Communications of the ACM, 43*(3): 107-114. March.

St. Amant, R., and Dudani, A. (2002). An environment for programming user interface softbots. *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI).* Pp. 119-122.

St. Amant, R., and Riedl, M. O. (2001). A perception/action substrate for cognitive modeling in HCI. *International Journal of Human-Computer Studies 55*(1): 15-39.

St. Amant, R., and Zettlemoyer, L. S. (2000). The user interface as an agent environment. *Proceedings of the International Conference on Autonomous Agents*. Pp. 483-490.

Zettlemoyer, L. S., and St. Amant, R. (1999). A visual medium for programmatic control of interactive applications. *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*. Pp. 199-206.