Position Statement and Overview: Sketch Recognition at MIT

Randall Davis
MIT Artificial Intelligence Laboratory
davis@ai.mit.edu

Position

The problem with software is not that it needs a good user interface, it needs to have *no* user interface. Interacting with software should — ideally — feel as natural, informal, rich, and easy as working with a human assistant.

One key to this lies in enabling means of interacting with software that are similarly natural, informal, rich and easy. We are making it possible for people involved in design and planning tasks to sketch, gesture, and talk about their ideas (rather than type, point, and click), and have the computer system understand their messy freehand sketches, their casual gestures, and the fragmentary utterances that are part and parcel of such interaction.

A second key lies in appropriate use each of the means of interaction. Our work to date has made it clear, for example, that different means are well suited to communicating different things: Geometry is best sketched, behavior and rationale are best described in words and gestures.

A third key lies in the claim that interaction will be effortless only if the listener is smart: effortless interaction and invisible interfaces must be knowledge-based. If it is to make sense of informal sketches, the listener has to understand something about the domain and something about how freehand sketches are drawn.

This paper provides an overview of six current pieces of work at the MIT AI Lab on the sketch recognition part of this overall goal.

Projects

Early Processing

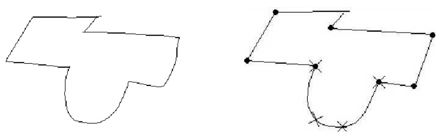
The focus in this part of our work is on interpreting the pixels produced by the user's strokes, producing low level geometric descriptions such as lines, ovals, rectangles, arbitrary polylines, curves and their combinations. This provides a compact representation and sets the stage for more abstract interpretation.

Our initial domain – mechanical engineering design — presents the interesting (and common) difficulty that there is no fixed set of shapes to be recognized. While there are a number of traditional symbols with somewhat predictable geometries (e.g., symbols for springs, pin joints, etc.), the system must also be able to deal with bodies of arbitrary shape that include both straight lines and curves. As consequence, accurate early processing of the basic geometry—finding corners, fitting both lines and curves—becomes particularly important.

Stroke processing consists of detecting vertices at the endpoints of linear segments of the stroke, then detecting and characterizing curved segments of the stroke.

Our approach takes advantage of the interactive nature of sketching, combining information from both stroke direction and speed data. We locate vertices by looking for points along the stroke that are minima of speed (the pen slows at corners) and/or maxima of the absolute value of curvature. To deal with the many false positives introduced by noise in the data, we rely on a variety of filtering techniques, including the use of a scale-space representation (for details, see [Sezgin01]).

An examples of the capability of our approach is shown below (input at left, parsed figure at right; dots indicate line endpoints, crosses indicate curve segment endpoints). Note that all of the curved segments have been modeled as curves, rather than the piecewise linear approximations that have been widely used previously.

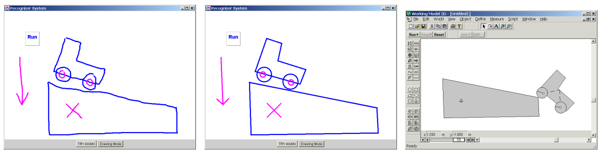


Object Interpretation

One important step toward sketch understanding is resolving ambiguities in the sketch, determining, for example, whether a circle is intended to indicate a wheel or a pin joint, and doing this as the user draws, so that it doesn't interfere with the design process. We have developed program that does this for freehand sketches of simple 2-D mechanical devices. (Another submission to this workshop describes the next generation of this system that we plan to build; here we describe the existing system).

The program is interesting in part because it employs a variety of knowledge sources to resolve ambiguity, including knowledge of drawing style and of mechanical engineering design, and because it "understands" the sketch in the sense that it recognizes patterns of strokes as depicting particular components. It illustrates that understanding by running a simulation of the device, giving designers a way to simulate their designs as they sketch them.

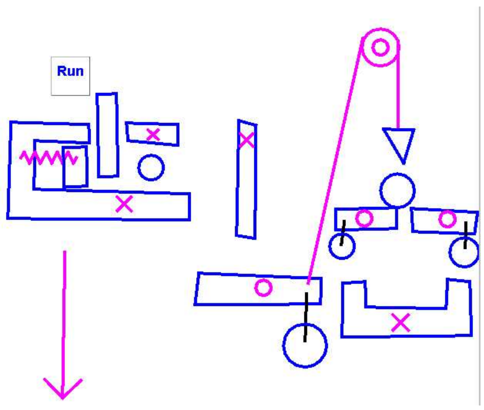
The figure below shows a session in which the user has drawn a simple car on a hill; the user's drawing as re-displayed by the program is shown in next. When the "Run" button is tapped, it transfers the design to a two-dimensional mechanical simulator which shows what will happen (at right). Despite ambiguities in



the sketch, the system is able to use context to select the correct interpretation, allowing the user to sketch without interruption.

Graphical Description of Behavior

The system above offers the ability to sketch structure, i.e., individual components and their connections. But the intended behavior of a device is not always obvious from its structure alone. Consider the (whimsical) egg-cracking device below, adapted from [Narayanan95]. The intent is that, as the stopper (the vertical bar near



the run button) is pulled up, the spring forces the ball to the right, it falls onto the see-saw, allowing the wedge to drop, cracking the egg into the frying pan. But if we simply run the simulation, nothing interesting happens: the stopper, responding to gravity, simply drops down a little, as does the ball, which then stays put. We need to be able to tell the system what is supposed to happen.

Designers routinely do this, explaining their designs to one another using sketches and verbal explanations of behavior, both of which can be understood long before the device has been fully specified. But current design tools fail almost completely to support this sort of interaction, instead forcing designers to specify details of the design by navigating a forest of menus and dialog boxes, rather than directly describing the behaviors with sketches and verbal explanations.

We have created a prototype system capable of interpreting multi-modal explanations for simple 2-D kinematic devices. The program generates a model of the events and the causal relationships between events that have been described via hand drawn sketches, sketched annotations, and verbal descriptions. This provides the user the ability to describe how the device should behave, and permits the system to make simple inference about the device (e.g., that the spring must be compressed if it is to force the ball to the right.

Sketching Software

Sketching is widely used in software design as a way of brainstorming, visualizing program organization, and understanding of requirements. We have developed a prototype sketch recognizer for UML (unified modeling language), a widely-used graphical notation for object-oriented systems. Within UML we focused on class diagrams, first because of their central role in describing program structure, and second because many of the symbols used in class diagrams are quite similar, and hence, they offer an interesting challenge for sketch recognition. (This system is the subject of another submission to the workshop.)

Describing Symbols; Learning New Ones

Our first-generation sketch recognizers relied on individual routines hand-written for each object to be recognized. We have now developed a language for describing the appearance of symbols, detailing the various geometric shapes and their relationships, with the goal that recognition routines will be automatically generated from these descriptions. An and-gate provides a simple example (Fig. 1).

As such descriptions are not trivial to write, we are also working on the problem of learning them from an example, using version spaces as a means of generating plausible hypotheses. To make the task as relatively painless, we want the system to learn the concept from as few examples as possible, ideally asking the user to drawn it only once, and thereafter have the system get the user's reaction to subsequent examples which it carefully generates with an eye toward reducing the space of possible descriptions.

An Electronic Drafting Table

Finally, we are working to incorporate a pen tracking device and a rear-projection display into an ordinary

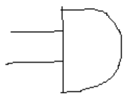
Sketch	Description	
	DEFINE AND-GATE line L1 L2 L3; arc A parallel L1 L2 same-horiz-position L1 L2 connected A.p1 L3.p1 connected A.p2 L3.p2	 meets L1.p2 L3; meets L1.p2 L3 meets L2.p2 L3 semi-circle A1 orientation(A1, 180) vertical L3

Figure 1: The description of an and-gate symbol.

drafting table, to allow designers to sketch in an environment that feels familiar, while still benefiting from the advantages of the electronic medium. This will be the basis for a variety of user studies to determine how "electronic paper" should feel.

Related Work

Previous work on freehand sketching (e.g., [Schneider88], [Eggli94]) has focused on more restricted kinds of input (e.g., spline curves). Our work on object recognition handles ambiguity in a more general way than previous work (e.g., [Do96], [Landay01]), which was typically restricted to disambiguating a single component. While other systems have provided behavior descriptions of devices, none have permitted the sort of graphical and gestural descriptions ours can handle.

References

[Do96] Do E, Gross M, Drawing as a Means to Design Reasoning, *AI and Design*, 1996.

[Eggli94] Eggli L, *Sketching with Constraints*, MS Thesis, University of Utah, 1994.

[Landay00] Long, Landay, Rowe, Michiels, Visual Similarities of Pen Gestures, *Proceedings of the CHI 2000 conference on Human factors in computing systems*, 2000.

[Schneider88] Schneider P, Phoenix: An Interactive Curve Design System Based on the Automatic Fitting of Hand-Sketched Curves, MS Thesis University of Washington, 1988.