Generating Domain Specific Sketch Recognizers From Object Descriptions

Tevfik Metin Sezgin and Randall Davis

The Problem: We use sketches as a medium for expressing ideas and saving thoughts. Sketching is especially common in early design as a means of communication, documentation and as a tool for stimulating thought. Despite the increasing availability of pen based PDAs and PCs, we still can't interact with our devices via sketching as we do with people. As a group, we are building a generic multi-domain sketch recognition architecture to make computers sketch literate. This sketch recognition system will differ from existing architectures in many aspects, including a language for describing shapes, mechanisms for learning new shapes, and a blackboard based recognition architecture with top-down and bottom-up recognizers. Here we describe a part of this system that generates efficient bottom-up recognizers by compiling object descriptions.

Motivation: As described in [2], current sketch recognition systems require users to hand-code individual recognizers as well as data structures for each object to be recognized. Some drawbacks of hand-coding individual recognizers are as follows: *(i)* writing recognizers and data structures is a labor intensive and error prone task, *(ii)* extending or modifying existing recognizers requires knowing how they work, *(iii)* because recognizers may be written by different programmers and may have different recognition algorithms, they lack a unified approach to recognition, *(iv)* users usually sketch parts of objects in a certain order and style, but current systems don't have a systemic way of exploiting this information to improve recognition accuracy and speed.

Previous Work: Current sketch based systems fall into the following categories: *(i)* systems with very limited recognition or systems that sidestep recognition to avoid problems induced by poor recognition ([3], [5]). *(ii)* systems that heavily depend on other modalities such as speech [7].

Previous work in our group has focused on sketch recognition [1]. One drawback of this approach was that adding new recognizable objects required writing new recognizers and data structures. This approach also suffered from the four problems mentioned above.

Approach: We aim to solve the problems mentioned above by automatically generating recognizers from object descriptions. Objects are described in an object description language that includes information such as components that form the object as well as the constraints that must be satisfied in order to have a legal instance of an object [4]. Given an object description, our system generates Java code that when compiled produces recognizers which function as knowledge sources in the larger blackboard based architecture. This automatic code generation scheme removes the need to manually write a separate recognizer for each object in the domain.

Our system reads object descriptions using a parser written using javacc (Sun's Java compiler compiler) and builds an abstract syntax tree. The AST has nodes for components, constraints of different types, and several other declarations such as renaming statements which provide a mechanism for referring to objects with a different name. The AST is then processed to generate Java code for individual recognizers. The generated code includes a series of nested for loops and conditional statements for cycling through available strokes in the sketching surface, generating permutations of strokes and testing if any of these permutations satisfy the requiredconstraints. If a particular subset of the strokes in the sketching surface satisfies all constraints, then a recognition is signalled to the blackboard. If only a subset of the constraints is satisfied then the blackboard is notified about a partial recognition.

The recognition algorithm outlined above has exponential complexity. The exponential nature of sketch recognition task is also discussedmentioned in [6]. Main cause of this exponential complexity is that the recognition algorithm assumes strokes forming an object can be drawn in any order and even in an interspersed fashion, thus we employ a brute force search for objects. As we found out in a user study, people sketch object components in certain styles (e.g., when drawing a stick figure, head is drawn first, left arm/leg is drawn before the right arm/leg). In addition we observed that people finish one object before they start drawing another. These key observations enable us to have polynomial time recognition algorithms by building HMMs to model different sketching styles. Training data for the HMMs is generated by encoding the output of early sketch processing toolkit described in [8]. We train a separate HMM for each object class. We use these models to segment and recognize sketches at the same time. This recognition scheme is more efficient compared to the brute force search method described above. Some of the results obtained using this method can be seen in Fig. 1.

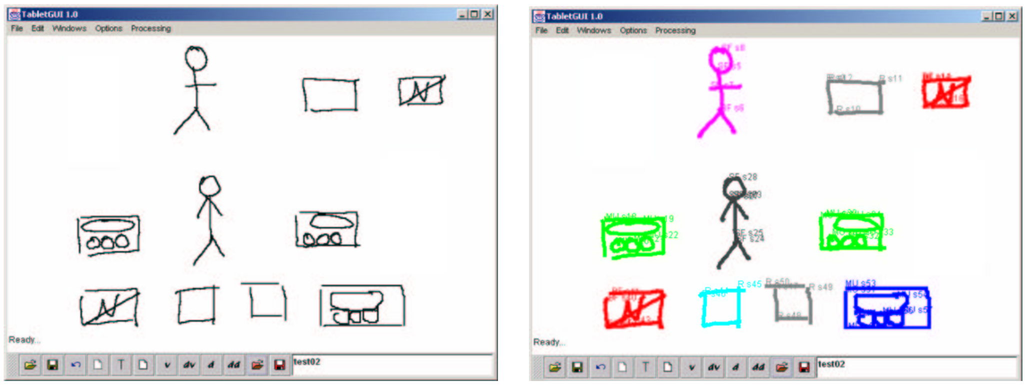
TabletGUI 1.0
File Edit Windows Options Processing
Ready.
test02
TabletGUI 1.0
File Edit Windows Options Processing
Ready.
test02

Figure 1: A number of hand sketched objects and the recognition results (on the right). Colors indicate object class. Note that our method can also detect different sketching styles within the same object class.

Impact: Along with the language described in [4] our system helps separating object descriptions from how an object is recognized and brings a unified approach to recognition. We also address the four problems mentioned before. For the first time we introduce a systematic way of exploiting stroke order and drawing styles for having robust and fast recognition.

Future Work: We described two sketch recognition methods. The code generation method exhaustively checks all possible stroke orderings and doesn't make assumptions about drawing order. On the other hand, the HMM based method utilizes users' sketching styles and runs in polynomial time. These two methods are complementary in that the if the user sketches in a style captured by the HMMs, the HMM based method is ideal. If the user sketches in a style that the HMMs haven't been trained before, the generated code can handle these cases. We are working on ways of combining these methods, as well as improving the speed and accuracy of these methods separately. We are also investigating how HHMMs can be used in recognition and segmentation.

Research Support: This research is supported by MIT's Project Oxygen.

References:

[1] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural sketch based interface. *Proceedings of IJCAI-2001*.

[2] Randall Davis. Designs for the future. *MIT Artificial Intelligence Laboratory Annual Abstract*, September 2002.

[3] M. Gross and E. Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of UIST 96*, pages 183–192, 1996.

[4] Tracy Hammond. Title to come.. *MIT Artificial Intelligence Laboratory Annual Abstract*, September 2002.

[5] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer, vol. 34, no. 3, March 2001, pp. 56-64.*, 2001.

[6] James V. Mahoney and Markus P.J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. *2002 AAAI Spring Symposium Series – Sketch Understanding*, September 2002.

[7] David R. McGee, Misha Pavel, Adriana Adami, Guoping Wang, and Philip R. Cohen. A visual modality for the augmentation of paper. In *Workshop on Perceptive User Interfaces*, November 2001.

[8] Tevfik Metin Sezgin. Feature point detection and curve approximation for early processing of f ree-hand sketches. Master's thesis, Massachusetts Institute of Technology, 2001.