Online Sketch Recognition from a Dynamic Perspective

by

Tevfik Metin Sezgin

Submitted to the Department of Electrical Engineering and Computer Science
as thesis proposal in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

April 2004

Author .....
Department of Electrical Engineering and Computer Science
April 02, 2004

Certified by .....
Randall Davis
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Online Sketch Recognition from a Dynamic Perspective

by

Tevfik Metin Sezgin

Submitted to the Department of Electrical Engineering and Computer Science
on April 02, 2004, as thesis proposal in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Sketching is a natural mode of interaction pervasively employed in a variety of settings. For example, people sketch during early design and brainstorming sessions to guide the thought process; when we communicate certain ideas, we use sketching as an additional modality to convey ideas that can not be put in words. The emergence of hardware such as PDAs and Tablet PCs has enabled capturing free-hand sketches, paving the road to making sketching an additional human-computer interaction modality. Despite these advances in pen based information capture, not enough effort has been put into using computers to understand and reason about sketches. So far, approaches to sketch recognition have treated sketches as static finished products or images and have applied vision algorithms for recognition. However, unlike images, sketches come to being in an incremental and interactive fashion and their processing should reflect this. In this proposal, we suggest two complementary sketch recognition techniques that take the incremental nature of sketching into account. The first performs a model analysis derive efficient recognition strategies that guides the correspondence search between a sketch and an object model. Our analysis takes into consideration that sketches are constructed one stroke at a time. The second technique we present is based on the observation that people often have a preferred order of putting the strokes on the sketching surface when they sketch. We propose to recognize what the user is most likely to be sketching by observing stroke drawing order. The two approaches proposed in this proposal act as complementary methods for efficient online sketch recognition.

Thesis Supervisor: Randall Davis
Title: Professor

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Sketching is a natural modality that is pervasively employed in a variety of settings. People sketch during early design and brainstorming sessions to guide the thought process and use sketches as a means of documentation[49]. When communicating certain ideas, we use sketching as an additional modality to convey thoughts that can not be put in words.

The increasing availability of pen based hardware such as PDAs and Tablet PCs makes capturing sketches easier than ever. Despite their ubiquity, suitability for certain HCI tasks, and the availability of supporting hardware, there is little computer support for sketches. We believe making computers sketch literate will produce smarter, more natural user interfaces. Noting the importance and potential impact of sketch recognition, several authors have proposed sketch recognition architectures and sketch based interfaces [30], [34], [41], [40], [46], [25], [35], [19]. These systems by and large either take a very simplistic approach to recognition (allowing only single stroke Graffiti-like input) or they try to apply to sketch recognition computer vision techniques that were developed to deal with static images. In this proposal, we first propose a model-based recognition technique that combines knowledge of prespecified object descriptions and the observations made during sketching to achieve conservative hypothesis generation during recognition. Next, we argue that rather than treating sketches as images, viewing sketching as an interactive and incremental process is not only more natural but also allows us to use a host of techniques that can take advantage of the dynamic properties of sketches such as stroke ordering, and timing information.

In the remainder of this chapter, we clarify what we mean by a sketch and define the sketch recognition task. In chapter 2, we briefly review the approaches taken to do model based object recognition and sketch recognition. In chapter 3, we discuss properties of sketches that distinguish them from images. In chapters 4 and 5, we propose two approaches to sketch recognition that utilize the dynamic nature of online sketching. We will conclude with our proposed evaluation methodology to test the performance of these algorithms.

Figure 1-1: Example showing what we mean by a sketch. Note the messy, freehand nature of the drawings.

1.2 Terminology

1.2.1 Freehand sketches

By a sketch, we mean informal, messy freehand drawings. These are unlike the clean, finished diagrams that have been the focus of graphics recognition community [1]. Because our focus is online sketch recognition, we represent sketches by a group of time-stamped strokes, each of which in turn is formed of a series of time-stamped points indicating the location and the time the pen was sampled.

1.2.2 Sketch recognition

The input to sketch recognition is a set of raw strokes. The output is groups of strokes labeled by their object type (i.e., a group of strokes labeled as a stick-figure). Strokes in each group are also labeled indicating what part of the object model they match (e.g., the fifth stroke in the group is the head of the stick-figure). We characterize the sketch recognition process in terms of three tasks:

- *Segmentation:* The task of grouping strokes so that those constituting the same object end up in the same group. At this point it is not known what object the strokes form. For example, in Fig. 1-1, the correct segmentation gives us four disjoint sets of strokes.

- *Classification:* Classification is the task of determining which object each group of strokes represents. For Fig. 1-1, recognition would indicate that the first object in the sketch is a stick-figure.

- *Labeling:* Labeling is the task of assigning labels to components of a recognized object (i.e., the head, the torso, the legs and the arms in the stick-figure in Fig. 1-1).

The three tasks described above are conceptually separate, but in a sketch recognition scenario, they may be intermixed. For example, it could be the case that a

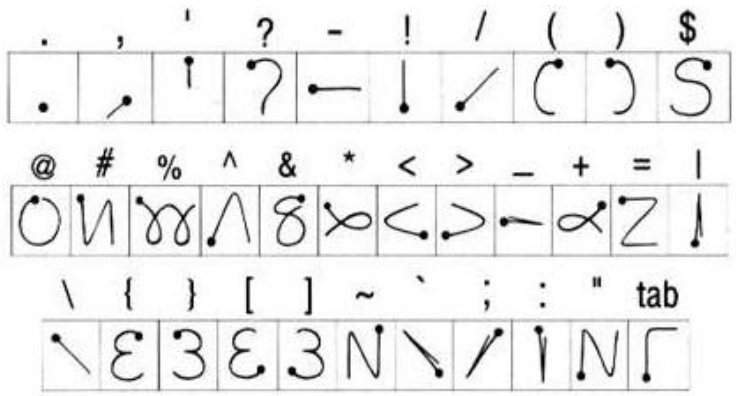

Figure 1-2: Gesture based Graffiti input scheme for PDAs. Input gestures must be drawn in a single stroke, starting at the heavy dot. Shapes of the some gestures do not look like the intended symbols (e.g., '%', '@').

particular recognition architecture classifies an object and labels its constituent parts simultaneously. More likely, segmentation is done first, followed by classification and labeling.

1.2.3 Sketching vs. Graffiti

Graffiti is a single stroke gesture recognition scheme developed mainly for PDAs. One of the most popular gesture recognition methods for Graffiti-like input is described by Rubine [44]. Fig. 1-2 shows the graffiti alphabet for punctuation and various symbols. In our view, Graffiti has a number of properties which make it unnatural and unlike sketching. In particular:

- Symbols are specified by gestures which do not necessarily resemble the target shapes (e.g. '%', '@' in Fig. 1-2).
- Input gestures must be drawn in a single stroke
- The gesture must start at a predetermined point (e.g., starting at the heavy dot as shown in the figure).
- The starting point may affect classification because some symbols are actually specified with the same gesture but with different starting points (e.g., [and { in Fig. 1-2).

Chapter 2

Review of the object and sketch recognition literature

Of the two methods we propose, the one in chapter 4 uses knowledge of object appearance provided by a model of object shape. This approach can be characterized as being *model-based*. Model-based approaches to computer vision have been extensively explored by the vision research community. Sketches have certain characteristics that make sketch recognition different from object recognition, but we find it useful to review the model-based object recognition literature to help us establish terminology and summarize what has been done in the field. Where appropriate, we point out aspects of model based vision relevant to sketch recognition.

2.1 Model-based recognition

2.1.1 Terminology

Model-based object recognition is characterized by the fact that the knowledge about the objects is given by an explicit model of shape [42]. In model-based object recognition, *models* describe objects in terms of primitives and relationships between primitives (constraints). In the computer vision literature, models are also referred to as *templates*, and model features are referred to as *template slots* or *slots*. Primitives of the model are called *model features*, and primitives forming the scene are called *scene features*. Model and scene features need not be exactly the same. For example, 3D model edges may appear as 2D edges in the scene, but they are related by some function (translation, rotation, projection). The scene features include primitives originating from the objects in the scene that we would like to recognize, and may also contain *spurious features* [1].

[1]Spurious features are features originating from unknown objects or from errors made at the low level processing stage (i.e., edge detection, feature point detection).

2.1.2 Object recognition

Recognition requires finding if any of the known objects are present in a given scene, and if so in what pose (i.e., location, scale, rotation parameters).

Pope [42] lists some of the common operations employed by recognition architectures:

- *Feature detection:* In the vision literature, features are usually primitives such as edges, circular, quadratic or cubic arc segments, extracted from the intensity image by lower level processing. In our case, the features are generated by the Sketching Toolkit [45]. Because we are dealing with online sketching and individual stroke information is available, unlike the results of edge detection algorithms employed in the vision literature, our methods return fewer *split* features (i.e., a line broken into smaller collinear lines).

- *Perceptual organization:* In this stage, detected features are grouped together based on perceptual grouping heuristics. This grouping is usually a domain independent process. As we discuss later, in the case of sketches, we believe the groups created in this phase can be used for indexing the models.

- *Matching:* Matching is the process of establishing a mapping between scene and model features, and determining pose. Approaches taken to do matching include searching the *correspondence space*, the *transformation space* and combination of the two. Correspondence space is the space of possible matches (i.e., space of pairings between model and image features). Transformation space is the space of possible object poses, viewpoints, or transformations between object and camera [42].

- *Indexing:* Indexing is the use of lookup-tables indexed by keys derived from feature groups to speed up recognition. Given a group of scene features, a key is computed from them. The table entry indexed by this key lists models that can potentially match the scene features along with additional pose and correspondence information. The goal of indexing is to form groups of features with richer constraints and use these groups to narrow down the set of candidate models to just those that contain these features. In chapter 4, we introduce methods for detecting unique constraint and component groups, and we propose to use these groups as a form of indexing to narrow down the list of candidate models. As we show in chapter 4, we believe performing a through analysis of object models can yield unambiguous starting-points for bootstrapping bottom-up recognition.

- *Verification:* Verification is the process of deciding whether the matches produced in the matching stage are indeed present in the picture. This problem has received less attention the literature than other stages in

2.1.3 Object recognition systems

In order to review approaches taken to object recognition and assess their complexities, we discuss specific approaches to matching and indexing. We also assess the suitability and limitations of using model-based object recognition techniques for sketch recognition. Where appropriate we will point out limitations of these techniques for sketch recognition.

Matching

In [23], Grimson gives an analysis of the combinatorics of the *interpretation tree* (IT) based object recognition method, with heuristics for pruning the depth-first-search once good enough matches are found. The used models are 2D descriptions of objects in terms of lines. Matching is done by building an interpretation tree in a depth-first fashion such that at each level l of the tree, the scene feature f_l is matched against model features, instantiating partial interpretations[2]. For each node in the interpretation tree at level l, the i^{th} outgoing branch corresponds to assigning the scene feature f_{l+1} to the i^{th} model feature and instantiating a new partial interpretation. A scene feature is matched against model features, and if there is no match, the scene feature can be matched against a special model feature called the *null character* or *wild card* feature. This is a way of allowing the matching to continue in presence of features that do not match the model (e.g. spurious features).

Grimson shows that if there are c object features uniformly distributed within the scene features s, then the size of the IT is polynomially bounded as long as $s \leq \frac{2m}{\kappa^2}$, where m is the number of model features (template slots) and κ is a parameter determined by the sensing system and object model. Typical values for κ are less than 1 except in extremely noisy settings. This implies that we can tolerate large amounts of spurious data and still be efficient in the case of sketching, because the Sketching Toolkit [45] returns far fewer spurious features than conventional edge detection algorithms do in typical 2D intensity images[3]. This method seems to have promising potential in efficient matching, though the analysis done by Grimson assumes uniform distribution of scene features.

In [17], Fisher gives a variant of the IT approach that doesn't use *null* branches. He suggests that by modifying the IT algorithm slightly, it is possible to increase speed by factors of 4-10. In a later paper [18], he compares ten variants of the original IT algorithm. He concludes that variations of the algorithm without null branches and those that use hierarchical representations of object models perform better. It is clear that from a theoretical point of view speeding up recognition by a constant factor is valuable, but its value in practice is not as clear.

Ullman's alignment algorithm mixes correspondence space search with transformation space search and uses *anchors*, groups of features sufficient for viewpoint determination, to determine viewpoint information.

[2] A partial interpretation is an assignment of scene features to a proper subset of model features.

[3] This is mostly because we have access to the stroke information rather than pixel level information.

In [9], Cass presents an approach that unifies correspondence space and transformation space search. He describes how data and model feature pairings can be mapped into the transformation space. Correct matches are found by looking for cells in the transformation space with the highest number of intersecting distinct data-model matches.

Among the approaches above, the interpretation tree approach is a correspondence space method. The alignment method and the method presented by Cass perform search in a unified space. Because sketches may not preserve affine properties, methods using the transformation space are not appropriate.

Indexing algorithms

In [39], Olson describes a probabilistic approach to indexing. The indexing is done over three dimensional transformations of three points using *the probabilistic peaking effect* observation. The probabilistic peaking effect is the observation that angles and length ratios for lines on a model do not change for most of the viewing sphere. This indexing scheme is used to test how well triples of image points match with triples of model points.

Costa and Shapiro describe a 3D object recognition system that makes use of indexing in [11]. They focus on recognition from CAD models. They use features such as ellipses, coaxial arcs, parallel pairs of lines, U, Z, L, Y and V junctions. In addition, they use relations such as *shares a line*, *shares an arc*, *coaxial*, *share two lines*, *close at external points*, *encloses, enclosed*. Two features and the relationship between them form a graph with one edge. A unique hash is assigned to each such subgraph. During the indexing phase, each object is characterized by its view models[4]. For each view model, the features and corresponding 2-graphs are computed, and a table indexed by a hash computed from 2-graphs is created with view-model labels as entries. In the recognition phase, 2-graphs in the scene are computed and scores are accumulated for each model indexed by the 2-graphs. Models with the highest scores are selected.

Stein and Medioni [47] introduce an encoding of polygonal models in terms of *super segments*. Super segments are characterized by properties such as arc length, angles between consecutive segments, location, orientation and eccentricity. They use super segments for indexing. The indexing scheme allows hypotheses to be retrieved from super segments present in the scene. Then mutually consistent hypotheses are clustered together and the pose estimate is refined. Their system also handles occluded objects relatively better.

Califano and Rakesh propose a high dimensional indexing scheme and claim that having higher dimensional descriptive global invariants produce better indexing [8]. The goal is to minimize overflow of the buckets, number of false positive, and sensitivity to quantization by using high dimensional descriptive global invariants. Their system uses a two step table lookup scheme where seven global invariants are computed correlating triplets of local curve descriptors. This is more of a theoretical

[4]View models are also called *aspects* in the literature.

paper. It is not clear how the ideas would work in practice, although they have a proof of concept example.

2.1.4 Automatic code generation in CAD based vision

Perhaps the earliest work in automatic code generation for model-based vision is by Goad[22]. The key observation is that some visibility calculations can be done offline, compiling the object models into recognizers. Projection and rotation operations are done by lookups performed on tables precomputed at compile time. Their depth first search algorithm has a three step approach:

- Predict: An edge is selected from the model and projected into the image space based on the current hypothesis.
- Observe: Image features near the projected model feature are found.
- Back-project The observed features are projected back to the model space and the current hypothesis is updated.

The algorithm presented by Goad also tries to maximize the chances of matching by trying to match features that are most likely to be visible, by choosing edges whose position can be most accurately determined and that will be most useful in determining the pose. All these decisions can be done at compile time.

In [28], Ikeuchi and Kanade describe a method for generating recognizers from geometric models. They define *aspects* as classes of poses that have the same topology for the visible faces. They use photometric stereo to obtain faces. Aspects are represented by the following features:

- Face moments
- Face relationships
- Face shape
- Edge relationships
- Extended Gaussian Image
- Surface characteristics distribution

Aspects are generated at compile time. Using this information, an interpretation tree is constructed. The interpretation tree consists of two parts: The first part classifies the input scene into one of the aspects; the second part is used to compute the exact pose. The IT uses aspect features to divide aspects into subsets. When the aspect is determined, the second part of the IT determines the pose using the following features:

- Center of mass of the Extended Gaussian Image distribution

- The position of observable region distribution
- Inertia direction of the original face, the original face shape
- Position of the surface characteristics distribution
- Position of the edges
- Position of the edges with respect to the position of the surface characteristics distribution

In runtime, the system uses three features:

- Edge map
- Needle map
- Depth map

From these, features used by the IT are derived, and object recognition is done by traversing the IT. The aspect classification is done by traversing the IT using 16 features (e.g., face area, inertia, face inertia ratio, number of surrounding edges). These features are ordered so that computationally cheaper ones are given priority. Nodes of the IT apply rules (among the 16) to guide the search process at runtime, but the choice of which rules to apply is made at compile time. The IT is pruned at compile time if there are any unnecessary branches. Once the aspect classification part of the IT is constructed, the pose determination part is generated. Finally, the IT is converted to an executable program in a straightforward fashion by implementing tree traversal as message passing between nodes of the tree. This is roughly equivalent to method invocation in Java.

One notable feature of [28] is that they model sensors explicitly during recognition strategy generation by computing what features would actually be detectable with a given set of sensors.

In [27], Hansen and Henderson describe automatic generation of recognizers from Computer Aided Geometric Design (CAGD) models. They present a method to rank features[5] based on their potential of constraining the pose and detectability. They use these features to construct a *strategy tree*.

The automatic feature selection is achieved by listing all features and then filtering them by *rarity*, *robustness*, *cost*, *completeness*, and *consistency*[6]. This is in contrast with the perceptually based description generation described in [51]. Depending on which order the filters are applied, the final feature set used to describe an object changes. A strategy tree is formed using the feature set. The top part of the tree determines the object and its pose using the features and the subtrees below that layer corroborate evidence to derive the pose. This paper is unclear on how pose is

[5]What they call feature is referred to as constraints in the MBV community.

[6]Completeness refers how well the features cover all views of the object. Consistency checks false positives due to lack of generativeness.

determined. It claims that layer one does object identification and pose estimation, yet the second part does the same thing. It seems like the purpose of the second layer is to confirm and strengthen the hypothesis generated in the first layer. The paper also presents a number of heuristics for building the second level of the tree.

In [5], Arman and Aggarwal describe a CAD-based vision system. They capture model information using 3D laser scanners. The captured data is converted into surface patches. This surface information is then used to create a *recognition tree*. Recognition strategy is compiled by ranking features by characteristics such as feature size, visibility, relationship to neighbors, distinctiveness, computational cost, reliability and frequency of occurrence. Matching starts with the highest ranked feature and its neighbors and continues with the remaining highest ranking features.

In [7], Byne and Anderson describe a computer vision system that uses geometric CAD models. Their system allows them to take advantage of appearance information specified using a language. An example of the kinds of appearance based information include the degrees of freedom that the object may have. For example, if the system is used to recognize cars, then appearance information can be the fact that cars are found on a plane and they have only one degree of rotational freedom. They use edge and surface information along with Hough transform for hypothesis generation. This paper doesn't compile models but is included here as a CAD based vision system.

In the graphics recognition community, Wenyin describes a system that can learn from a small number of examples and perform recognition using a generic object recognition scheme [53]. The object recognition algorithms starts by matching a model feature to a data feature. The search continues by getting directions in which the search can be extended. Next, components in the computed directions are sorted by their distance to the partial match and considered for extension in that order. Object descriptions are read at runtime.

As the analysis above indicates, the most important feature of CAD based vision systems is the analysis of object models offline. In chapter 4, we discuss the kinds of analysis that can be done to achieve efficiency for online sketch recognition.

2.1.5 Subgraph isomorphism and graph algorithms

Subgraph isomorphism is a matching technique frequently employed in model-based vision. Object models and scenes are represented by graphs with nodes corresponding to features and edges representing spatial or geometric relationships between features. Recognition is achieved by finding subgraphs of the scene graph isomorphic to the object models. Because of their popularity in model-based vision, subgraph isomorphism algorithms have received much attention. Below we describe approaches to solving the subgraph isomorphism problem.

In [12], DePiero and Krout give a subgraph isomorphism algorithm which uses counts of length-r paths in a graph. The algorithm dynamically adjusts the 'horizon' associated with a comparison. The algorithm is approximate and runs in polynomial time.

Cordella et al. describe a graph matching algorithm with reduced memory requirements in [10]. They also mention that converting the recognition task to a

graph matching problem may cause some of the semantics behind the graph to be lost. This claim begs more elaboration.

In [38], Messmer describes algorithms for polynomial time graph isomorphism. In later work, Messmer and Bunke developed similar algorithms for polynomial time error correcting graph isomorphism in [37].

The methods above suffer from exponential running time or storage requirements, because subgraph isomorphism problem is NP complete. There have been efforts to develop approximate algorithms with polynomial time complexity. In [3], Almohamad and Duffua describe an approach for weighted graph matching that uses linear programming techniques. This is an approximate algorithm, and has polynomial time complexity. Their algorithm is for weighted graphs and it is not clear how it would work for the kinds of graphs encountered in sketch recognition. Another approximate solution to graph matching is presented by Umeyama [50]. The method is technically involved, and its practical utility is not clear.

Methods discussed above address subgraph isomorphism in general. Unlike the general case, planar subgraph isomorphisms can be found in polynomial time. Eppstein [16] describes an algorithm for solving planar subgraph isomorphism in polynomial time. These algorithms are applicable in problems where constraints between features are limited to a small neighborhood such that graphs remain planar. In general graphs induced from object descriptions of the kind proposed in [26] are not planar.

Another limitation of the subgraph isomorphism based recognition approaches is their inappropriateness for dynamically changing graphs. In sketching, although the object graphs remain static, the scene graph keeps evolving. In the graph theory literature, the term *dynamic graph algorithms* is used to refer to algorithms that are built to work with dynamically changing graphs through edge/node insertions and deletions [14]. To our knowledge, no dynamic algorithm has been devised for subgraph isomorphism.

2.2 Review of the sketch recognition literature

Sketch recognition is a relatively new research area. Although there were attempts to make computers sketch literate as early as the 1970's, the field is still relatively young. Among the current systems, there is much variation on many dimensions including:

- *Recognition scope*
- *Approaches to solving the problem*
- *Robustness of the approach*
- *Computational requirements*

This section reviews the sketch recognition related literature with the characteristics above in mind, allowing us to position our work in the larger context. We start with systems that sacrifice larger recognition scope for robustness and move towards more ambitious systems that try to recognize more objects, but are more brittle.

2.2.1 Sketching systems with gesture-like input

A gesture is a single stroke input drawn in a prespecified way. Graffiti, described in section 1.2.3, is one example. Sketch based systems that have been built with more emphasis on robustness and practical usability are usually gesture-based, because of the availability of robust gesture recognition methods.

SILK [30], is an example of such a gesture-based system. SILK allows users to specify the layout of GUI components using gestures. The system also supports grouping sketches of several GUIs to create a storyboard, but here we focus on the sketch recognition features of this system. The authors of SILK prefer usability by real users over supporting tasks that require a larger set of objects to be recognized. This is reflected on their choice of GUI design as their domain, and their use of a gesture-based approach for recognition. As illustrated by this system, a GUI design system can be built with a fairly small vocabulary. SILK recognizes four primitive components (rectangle, line, squiggly line, and ellipse). The system also recognizes seven basic GUI components (e.g., buttons, text boxes etc.) by interpreting how the primitive components are related. For example, a squiggly line drawn inside a rectangle is recognized as a button with a text label.

ISID [6] is another gesture-based system, a sketch based front-end for a CAD system. ISID uses gestures to specify editing operations. It allows the users to build CAD models by sketching raw strokes, and then applying operations (e.g., extrusion, subtraction). The system also supports some beautification by detecting and enforcing parallel, orthogonal, same-length and connection constraints.

Other gesture-based sketching systems include the Music Notepad by Zeleznik [20], and the "Cocktail Napkin" by Gross [24].

A sketch recognition mechanism based on a pattern representation scheme is discussed in [29] and [52]. The authors first introduce a pattern representation scheme using data compression (PRDC). The PRDC framework makes sequence compression and classification possible. Given training data, the input strokes are encoded using a dictionary based on the movement direction of the pen. The dictionary includes entries for each of the four cardinal directions (east, west, north, south), the four halfway directions, and their combinations. This results in the quantization of pen direction. Then the PRDC scheme is used to classify isolated object instances, which is a relatively simpler task compared to recognizing an object in presence of many others. Although the authors report promising initial results, because their test and training data is rather small, we believe this method needs a more thorough evaluation. Also due to the particular encoding scheme employed, this technique suffers from the four disadvantages that Graffiti based recognition architectures suffer from.

2.2.2 Model-based approaches to recognition

Mahoney and Fromherz describe a sketch recognition scheme based on subgraph matching in [34], [33], and [32]. Unlike the approaches above, which are gesture-based, their method is based on structural matching using subgraph isomorphism.

For each object to be recognized, an attributed model graph[7] is generated based on a syntactic description of the object that has previously been specified by the user. In this attributed graph, nodes correspond to the lines and the edges correspond to the geometric properties of the underlying lines and line relations. Given a sketch, the algorithm first generates an attributed graph representing the scene using the set of lines in the sketch and their relationships. Next, the scene graph is augmented by subgraphs corresponding to possible ambiguities in the interpretation of line relations. This augmentation operation – called *graph elaboration* – applies perceptual organization principles such as good continuation and proximity. Next, the algorithm looks for instances of the model graph in the scene graph by performing subgraph matching. This approach to sketch recognition can be thought of as an adaptation to sketches of the subgraph isomorphism approach, popular in computer vision, in a way that deals with the free-hand and noisy characteristics of sketches using perceptual organization principles. On the other hand, because the subgraph isomorphism treats the sketch as a static image, this approach does not capitalize on the dynamic properties of sketches such as the drawing order of strokes.

Shilman et al. describe a statistical, grammar based ink parsing algorithm for sketch recognition [46]. In this framework, the system designer describes objects with a declarative grammar and gives labeled training examples. The initial grammar requires predetermined constants for computing constraints between objects. A parser is generated using this information. During runtime, the parser looks at the sketching surface and generates a parse tree whose nodes correspond to alternate interpretations of the sketch. Next, Bayesian networks corresponding to interpretations at the parse tree nodes are generated. Most likely interpretations are chosen and expanded further, pruning nodes with probabilities less than a preset threshold. The authors report that their method outperforms a hand-coded parser for a domain with a relatively simple grammar. Having been inspired by language parsing, this approach is not built with the dynamic and interactive nature of sketching in mind. We believe attending to the dynamic aspect of sketching can be used to devise techniques which are better suited for online sketch recognition.

The sketch recognition system being developed in our group consists of:

- A language to for describing objects [26],
- A perceptually based learning system for learning shape descriptions [51],
- A blackboard based architecture with a top-down recognition component based on dynamically constructed Bayesian networks that allows recovery from bottom-up recognition errors[4],
- Multimodal components that support speech [2] and hand gestures [13].

The work described in this thesis complements the above sketch recognition framework by providing recognition mechanisms that take advantage of the incremental, dynamic nature of online sketching.

[7] An attributed graph is a graph whose nodes and edges can have attributes (e.g., edges or nodes can have types or numeric values associated with them).

In the next chapter, we discuss static and dynamic properties of sketches and lay out the motivation for our approach to sketch recognition.

Chapter 3

Properties of sketches

3.1 Static properties of sketches

Static properties of sketches are the properties that would be present even in a scanned image of a sketch. They include image-like properties such as digitization noise and others that are specific to sketches, such as messiness:

- Noise: Noise arises from digitization performed by a digitizing tablet, a scanner, or another imaging device. Each device has an inherent resolution and noise characteristic. This property of sketches also exists in images.
- Messiness: Messiness refers to phenomena such as overshooting, undershooting, messiness in the lines and jitters due to hand tremor (e.g., Fig. 3-1).
- High variability: Sketches are highly informal. Instances of the same icon may have varying aspect ratios and scales. Furthermore, parts (subcomponents) of an object may have varying aspect ratios within the object. This high variability rules out popular transformation space search approaches that rely on affine properties of images.
- Iconic: The sketches we deal with are mostly iconic (e.g., a triangle above a square for a house, a "light bulb" representing an idea, or a stick-figure). Their iconic nature makes it possible to have symbolic descriptions of sketches. There are domains with highly non-iconic properties, such as "police sketches" or artistic renderings.
- Compositional and hierarchical: Most sketches can be broken down into compositions of simpler objects, and those objects can be further broken down into primitives such as lines, circles etc. For example, a simple sketch of a car can be broken down into the body, and the wheels, and the wheels can be further broken down into two circles representing the wheel and the axle.

Figure 3-1: Example showing what we mean by a sketch. Note the messy, freehand nature of the drawings.

3.2 Dynamic properties of sketches

One of the advantages of sketching is that a sketch can be captured as it is constructed, making it possible to capture properties that may potentially aid recognition. Our stroke-based representation allows us to access stroke level information including what order the strokes came in, as well as timing/pressure/pen tilt information for individual points.

3.3 Observations about the sketching process

- Sketching is incremental, with strokes put on the sketching surface one at a time.
- Sketch recognition is interactive, because there is a two way communication channel: information flows from the user to the computer via the strokes drawn and the editing operations; and from the computer to the user in terms of computer's interpretation of the strokes and editing operations. This interactive nature of sketching is an important source of knowledge when it comes to confirming the correctness of a system's interpretation. For example, the longer the user lets an interpretation exist, the more certain the system can be about its interpretation [4].
- During sketching, objects are typically drawn one after the other. Although there are cases where people start sketching a new object before they complete the previous one, people almost invariably draw objects one at a time.
- Finally, sketching is a highly stylized process; people have strong biases in the way they sketch. We conducted user studies to validate this observation, and they in turn form the basis for the approach to recognition and segmentation that we present in chapter 5.

Because sketching is incremental, most of the time the sketching surface will have a partially drawn object. Partially drawn objects are a common source of ambiguity in sketches. One of the challenges in sketch recognition is to balance the tendency to generate all plausible hypotheses and the need to wait until we have enough components so the search is more constrained. In the next chapter, we propose methods to achieve this without assuming that users sketch in a stylized way or one object at a time. In chapter 5, we show how making these two assumptions can be used to formulate faster recognition algorithms.

Chapter 4

Efficient recognition using object models

As explained in the previous chapter, the incremental nature of sketching requires being conservative in creating partial interpretations. Here we propose methods to generate efficient recognition schemes from object descriptions provided in the language in [26]. These recognition schemes take advantage of a priori knowledge about the shapes in a given domain. First, we introduce the continuous sketch recognition concept and point out its difference from recognition of objects in images. Next, we present results from an experiment we conducted to explore the effects of reordering incoming strokes on the cost of recognition. Drawing on the observations of this experiment, we propose two techniques for efficient recognition by exploring the search space conservatively.

4.1 Continuous sketch recognition

From a blank sheet to the end of the sketching process, the sketching surface sees a number of plausible scenes formed of completed objects even if the drawing is not complete. In many circumstances, the recognition system may be required to recognize valid completed objects in the scene even if the sketch is not completed. One obvious scenario where object recognition is needed as the sketch is being constructed is when the designer of the sketch-based interface wants the system to show its understanding by displaying iconic descriptions or neatened versions of the objects that it recognizes. Another instance where it is required to recognize objects before a sketch is completed occurs in the case of editing. For example, in the domain of digital logic circuit sketches, if the user is sketching an RS flip-flop circuit composed of two $NAND$ gates and wires, the editing behavior of the sketching interface may depend on its operand (e.g., when the eraser part of the stylus is used on wires, we might want to delete parts that it touches; when used on the gates we might want to delete the whole gate).

Approaches to model based vision were developed with the assumption that all the scene elements are present. The matching algorithm we will describe in this chapter

is a variant of the interpretation tree algorithm that explores the search space in a breadth first fashion and deals with the incremental nature of sketching by generating hypotheses based on knowledge extracted by offline analysis of all the object models.

4.2 Effects of reordering component assignments

We now describe how a variant of the popular interpretation tree (IT) algorithm – a correspondence space search algorithm – performs for continuous sketch recognition[1]. Our goal is to give empirical results on the performance of a matching algorithm for real data, and motivate the approach we propose for achieving faster recognition.

4.2.1 An example: Recognizing stick-figures

We described the interpretation tree algorithm in section 2.1.3. Our experiments used a variant of this algorithm modified for continuous sketch recognition. The basic idea is to instantiate plausible partial interpretations of a given scene. The list of plausible partial interpretations is extended as new strokes are drawn. After each stroke is drawn, it is classified as a geometric primitive (line, polyline, oval, curve) using the early sketch processing toolkit described in [45]. Partial interpretations are created and updated. For each type of object to be recognized:

- If there are no partial interpretations for the object we are trying to recognize, and if the geometric primitive derived from the latest stroke fits into a slot[2] without violating any constraints, create a new, partially instantiated template with that primitive assigned to that slot.
- If there are existing partial interpretations that can be extended with the latest primitive without violating any constraints, these interpretations are cloned and extended.

4.2.2 Experiments

We implemented a stick-figure recognizer to determine how the simple recognition strategy described above performs for continuous sketch recognition. Because sketches are created incrementally and the drawing order for parts of a figure can change, we tested how many partial interpretations we get for different drawing orders. We recorded raw strokes for a stick-figure (total of 6 strokes) and added the strokes to the sketching surface in different drawing orders to simulate different ways in which an

[1] As mentioned earlier, because sketches have a high degree of variability (e.g., non-affine scaling properties), transformation space search becomes inappropriate for recognition. We thus focus on correspondence space methods which try to find correspondences between image features and model features subject to constraints.

[2] By a slot, we refer to a component of a particular object model. For example, a plus sign has two slots corresponding to the intersecting horizontal and vertical lines. This is referred to as object feature in the computer vision literature.

object can be drawn. To measure the cost of recognition for a particular ordering, we used the number of partial interpretations that were instantiated at the completion of the sketch. Fig. 4-1 shows the costs for different orders sorted by cost.

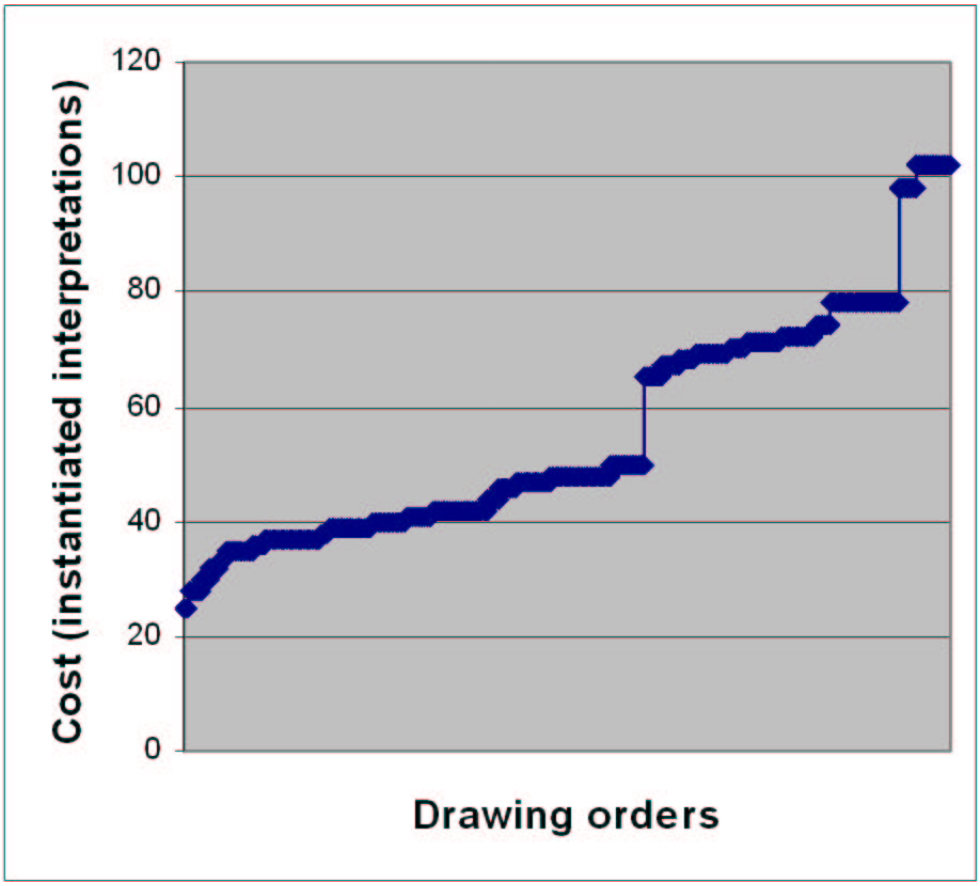

Figure 4-1: Number of partial interpretations generated during recognition of stick-figures drawn with different stroke orderings, sorted in ascending order. The y-axis shows the cost.

4.2.3 Observations

As seen in the figure, costs for different orders range from 24 to 121. The reason for this difference is that depending on how an object is drawn, the number of partial interpretations will vary. In other words, the branching factor of the corresponding IT will be different for different drawing orders. An example illustrates why this happens: For a stick-figure, if we start with two touching lines, they could potentially be pairs of arms, legs, or the body and any other limb. On the other hand, if we have an oval touching a line, these strokes can only be the head and the body. In one case, there are multiple ways in which two strokes can be labeled, while in the other the labeling is unique.

One drawback of the interpretation tree algorithm is that levels in the tree correspond to unique scene elements. In order to have a framework where object com-

ponents can be filled in a preferred order, from this point on, we assume a variant of the IT algorithm where each level of the IT is associated with a model feature, and branches correspond to assigning different scene features to the model feature in question.

4.3 Improvements through object model analysis

In this section we list speed improvements that can be achieved by an analysis of object descriptions. The first four techniques are based on methods of efficiently solving constraint satisfaction problems.

4.3.1 Most constraining variable first

One of our goals during recognition is to build an interpretation tree that quickly prunes out paths corresponding to infeasible labelings. Because the likelihood of a given set of features accidentally satisfying a constraint is inversely proportional to the number of features participating in that constraint, we suggest giving higher precedence to constraints involving more components. Given a scene, if we assume the probability of different constraints being accidentally satisfied is independent, then giving precedence to constraints that are less likely to be accidentally satisfied minimizes the expected size of the interpretation tree.

4.3.2 Variable with the smallest runtime domain first

Another technique for keeping the search space under control is to keep a low branching factor. In our case, branches correspond to scene features. Given a scene feature, the model features it can fit are constrained by the type of the primitive and the type of the model feature. The number of primitives that can be matched against a particular model feature is determined during the sketching session.

For example, if we have an object (like the stick-figure) described by lines and circles, and, if we have many more circles on the sketching surface than there are lines, it may be preferable to match the body and the limbs before the head.

4.3.3 Cycle cut-set removal

Cycle cut-set removal is a method used to reduce the amount of backtracking needed in a constraint satisfaction problem (CSP). Given the primal graph[3] for a CSP, the cycle cut-set removal method assigns values to nodes that make the CSP graph acyclic (i.e., a tree). The set of nodes making the graph acyclic is called the *cycle cut-set*. As an example, see the primal graph for the stick-figure model shown in Fig. 4-2. There are multiple ways in which this graph may be converted into a tree. For example,

[3]The primal graph for a CSP is a graph with nodes representing variables and edges for each constraint.

removing the body, the left leg and the left arm, or the right arm and the right leg would turn the graph into a tree.

The issue of which cut-set to prefer is an important one and can be answered using two criteria. Cut-sets with fewer nodes are preferable because removing larger cut-sets require more variable instantiations or equivalently more assignment combinations.

Alternatively, the choice of which cut-set to remove can be based on the expected combinatoric cost of assigning values to the nodes in the cut-set. Most of the time, the minimal cut-set of a graph will include subgraphs that are not related by many constraints[48], making it costly to assign values to the nodes in the cut-set. The expected combinatoric cost measures how ambiguous a cut-set is, as outlined in section 4.3.5. The goal would be to choose a cut-set where nodes (variables of the CSP) are related with constraints that allow an unambiguous assignment of values to the nodes. For the stick-figure example, removing the head and the body removes all the cycles in the graph based on the observation that labeling of the components in the cut-set can be done unambiguously.

Finding the right cut-set can be an expensive combinatoric operation, but this cost will be incurred offline.

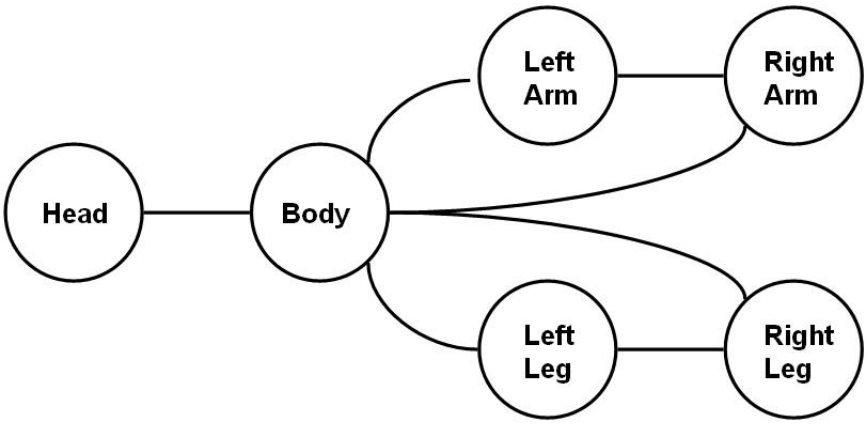

Figure 4-2: The constraint graph for the stick-figure.

4.3.4 Recognizing k-trees

If the primal graph of our object description forms a *k-tree*, then a consistent labeling can be found efficiently using results from constraint satisfaction problems. A k-tree is defined recursively as:

- a complete graph with k nodes, or
- a tree where we can find a node n that has k neighbors forming a complete graph such that when n is removed, the new graph is a k-tree

Verification of the k-tree property of a graph can be done in linear time. The verification process also returns the appropriate ordering for labeling model features.

```
(define Stick-Figure
 (components
   (Ellipse head)
   (Line body)
   (Line leftArm)
   (Line rightArm)
   (Line leftLeg)
   (Line rightLeg))
 (constraints
c0   (meets leftArm rightArm leftArm.p1 rightArm.p1)
c1   (meets leftLeg rightLeg leftLeg.p1 rightLeg.p1)
c2   (left leftArm     rightArm)
c3   (left leftLeg     rightLeg)
c4   (touches leftArm  body leftArm.p1)
c5   (touches rightArm body rightArm.p1)
c6   (meets leftLeg    body leftLeg.p1 body.p2)
c7   (meets rightLeg   body rightLeg.p1 body.p2)
c8   (touches body     head body.p1)))
```

Figure 4-3: Stick-Figure and the object model.

4.3.5 Detecting and exploiting unique constraint groups

An interesting question to ask is what component or constraint groups provide sufficient information for labeling with no or little ambiguity. For instance, in the case of the stick-figure example, the existence of a circle is good enough information for labeling it as the head, or the existence of a line touching a circle is enough for labeling the circle as the head and the line as the body, while having two connected lines is not enough information to label them as the left and right arms.

Our goal is to see if a set of constraints $c_0, c_1, ..c_{n-1}$ could have possibly been satisfied by one or more scene features whose true labeling L is different from those suggested by the satisfied constraints. We answer this question by assuming that the sketching surface has primitives satisfying constraints $c_0, c_1, ..c_{n-1}$. Primitives are given anonymous labels such as $line_0, line_1...$, $circle_0, circle_1....$ Then we compute the satisfied constraints and the implied labelings in the object model. If the set of labelings include labelings other than L, that means the constraints $c_0, c_1, ..c_{n-1}$ do not impose an unambiguous labeling of primitives, and it is preferable to wait for more primitives to be drawn by the user. This implicitly imposes a preferred filling order over the object features.

We can treat the type of an object feature as a unary constraint on it, enabling this approach to detect cases where simply having a primitive of the right type suffices to do the labeling. For instance, in the stick-figure example, if we view the type of the head as being a unary constraint (i.e., interpret `(Circle head)` as a unary constraint) on the primitive, then we can anonymize this constraint and check if this constraint alone is sufficient for labeling. As it can be easily verified, in this case, type information alone is sufficient for labeling.

Uniqueness of constraint sets can be checked against other object models as well. This would test if it is possible for parts of other objects to satisfy constraints of this model.

4.3.6 Using groups and hierarchy for recognition

The language specified in [26] allows hierarchical descriptions and groupings, although this kind of information may not always be available. When hierarchy and grouping information is present, it is natural to recognize subgroups and sub-shapes separately.

4.3.7 Using planar subgraph isomorphism

As mentioned in chapter 2, one approach to object recognition is to generate attributed scene and model graphs and look for subgraphs of the scene isomorphic to the model graph. These methods are costly because subgraph isomorphism is NP-complete. On the other hand, planar subgraph isomorphisms can be found in polynomial time. It may be possible to make simplifying assumptions about object models and the scene so that planar subgraph isomorphism can be used for fast recognition.

One possible avenue to pursue in this case would be to analyze object descriptions and choose a set of constraints with corresponding attributed graphs that are planar. The challenge in this case will be to choose a subset of constraints that is highly discriminative (i.e., resulting in few false positives). This can be done by analyzing the object description. The analysis would employ a two step generate and test approach where the generation stage constructs subsets of constraints from the object description such that the corresponding attributed model graph is planar. The testing stage would assess the discriminative power of the chosen constraints by checking how many false positives it allows given the knowledge of all object models.

Another way of choosing constraint subsets would be to give precedence to constraints that are known to be more important from a perceptual aspect (e.g., the meets constraint is perceptually more important than relative length).

4.3.8 Using connectivity information from polygons

The general recognition strategy we described in section 4.2.1 assigns primitives to model features. Because polylines are not in the set of primitives defined by the language in [26], our recognition strategy breaks polylines into straight line segments and performs recognition from lines. There are three ways in which we will use this information:

- Having lines derived from a polyline makes the the `connected` constraint between the lines absolute. This helps in reducing some of the ambiguities resulting from the noise in computing low level spatial relationships.

- Because different objects in drawings rarely share polylines, we can enforce the constraint that an object model either use all the lines descending from the polylines, or not include any. Using this information will allow us to rule out hypotheses where the object model contains fewer lines than there are in an unmatched polyline.

- If a stroke is classified as a polygon (i.e., closed polyline) by our low level sketch processing toolkit[45], then it can only match those models where line segments form a closed shape. We plan to use this information to narrow down possible lists of models to match against data. In addition, we plan to compile lists of model features forming closed paths for each object model offline, and try to match polygons against these primitive sets.

Chapter 5

Efficient recognition by exploiting sketching styles

Current sketching systems are indifferent to who is using the system, employing the same recognition routines for all users. But user studies we have conducted indicate clearly that different users have different sketching preferences and styles, and that there is considerable value in being able to capture these different styles. The framework we introduce in this chapter provides a mechanism for capturing an individual's preferred stroke ordering during sketching, and uses it for efficient sketch recognition. We show how viewing sketching as an incremental and interactive process provides extra leverage over computer vision methods developed for performing recognition in static scenes.

5.1 User study

We ran user studies to assess the degree to which people have sketching styles, by which we mean the stoke order used when drawing an item. For example, if one starts drawing a stick-figure with the head, then draws the torso, the legs and the arms respectively, we regard this as a style different from the one where the arms are drawn before the legs. Our user study asked users to sketch various icons, diagrams and scenes from six domains. Example tasks included:

- Finite state machines performing simple tasks such as recognizing a regular language.
- Unified Modeling Language (UML) diagrams depicting the design of simple programs.
- Scenes with stick-figures playing certain sports.
- Course of Action Diagram symbols used in the military to mark maps and plans.
- Digital circuit diagrams that implement a simple logic expression.
- Emoticons expressing happy, sad, surprised and angry faces.

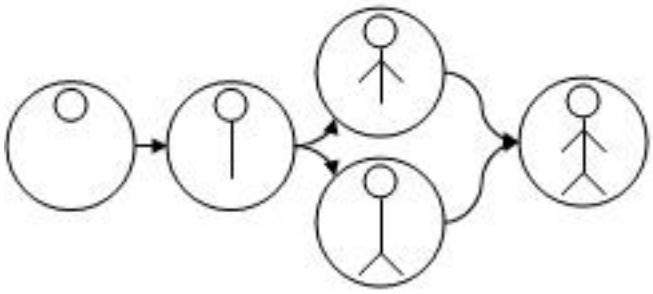

Figure 5-1: Sketching style diagram for a stick figure illustrating two different ways of drawing stick figures: In both cases the stick figure was drawn using four strokes, but the order in which the legs and arms were drawn is different.

We asked 10 subjects to draw three sketches from each of the six domains, collecting a total of 180 sketches. Requests were given to subjects in an order that interspersed domains to reduce the correlation between sketching styles within a domain. Sketches were captured using a digitizing LCD tablet.

We analyzed the sketches by constructing sketching style diagrams for each user, one for each object in our domains. *Sketching style diagrams* provide a concise, graphical way of representing how different instances of the same object were drawn. Nodes of a sketching style diagram correspond to partial drawings of an object; nodes are connected by arcs that correspond to strokes. Fig. 5-1 illustrates the sketching style diagram of a particular user for the stick figure example described above. Our inspection of the style diagrams revealed that:

- People sketch objects in a highly stylized fashion. In drawing the stick figure, for example, one of our subjects always started with the head and the torso, and finished with the arms or the legs (Fig.5-1).
- Sketching styles for individual users agree across sessions.
- Subjects preferred an order (e.g., left-to-right) when drawing symmetric objects (e.g., the two arms) or arrays of similar objects (e.g., three collinear circles).
- Enclosing objects were usually drawn first (e.g., the outer circle in happy faces).

The user study confirmed our conjecture about the stylized nature of sketching. In order to capitalize on this structure we have used Hidden Markov Models (HMMs) to model different sketching styles. HMMs are appropriate for this task because sketching can be seen as generating a sequence of strokes (observations), and HMMs have successfully been used to analyze time series data. Next, we describe why we use HMMs, briefly review HMMs and explain how we applied them to the problem at hand.

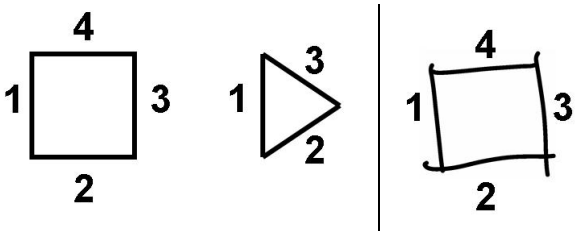

Figure 5-2: Symbols for *stop* and *play* (on the left), and a sketched *stop* symbol (right).

5.2 HMM-Based Recognition

5.2.1 The intuition

To give an intuitive explanation of why and how we use HMMs for recognition, we consider an over-simplified sketch recognition scenario with only two types of objects: *play* and *stop* symbols as shown in Fig. 5-2. Assume that the user draws the symbols always using the same stroke ordering, indicated by numbers in the figure. Our task is to recognize which of these objects is present in a given scene that is known to contain only a single instance of one of these objects.

Suppose the user draws the *stop* symbol as shown in Fig. 5-2. Assuming we can reliably recognize the individual strokes as lines, we can use the following criteria based on the drawing order to do recognition:

- If the user drew a vertical line, followed by horizontal, vertical and horizontal lines, then the object in the scene must be a *stop* symbol.
- If the user drew a vertical line first, followed by positively sloped line and finished with a negatively sloped line, then the object in the scene must be a *play* symbol.

In effect, the above approach works by encoding the user input (in terms of horizontal, vertical lines etc.), generating a *sequence of observations* describing the scene, and comparing this observation sequence to its model of how the user is known to sketch. The result of the comparison is binary, indicating either we have a match or not. This toy example provides insight into how stroke ordering can be used for recognition in an over-simplified scenario. For real sketches, we would like a framework meeting the following requirements:

- **Support for multiple classes and compact representation of drawing orders:** We should be able to recognize multiple classes of objects. We also want to accommodate multiple drawing orders instead of just one. For example, if 20% of the time the user sketches the *stop* symbol starting with a horizontal line, this should be accounted for by the recognition procedure.
- **Ability to handle variations in drawing and encoding length:** Users should be able to draw freely. For example, they should be allowed to draw the *stop* symbol using three strokes instead of four (thus generating an encoding of the sketch with only three observations instead of four).

- **Probabilistic matching score:** We would like the result of matching an observation sequence against a model to be a continuous value reflecting the likelihood of using that particular drawing order for drawing the object. This is required if we are to have a mathematically sound framework for combining the outputs of multiple matching operations for scenes with multiple objects such that, among plausible interpretations, those corresponding to more frequently used orders are preferred, as we explain later.

- **Learning:** In practice, different drawing orders will have similar subsequences. Ideally the system should allow learning compact representations of drawing orders from labeled sketch examples. These representations should incorporate knowledge of relevant drawing orders and how frequently they occur.

- **Ability to handle continuous observations:** The method should be extensible such that, if needed, it can use continuous observation distributions as opposed to a set of discrete symbols.

In summary, the requirements above imply the need for a systematic way of measuring how well observation subsequences (discrete or continuous) match individual object models, as well as a well-founded method for learning multiple drawing orders for objects. We use HMMs for this purpose, because HMMs provide a well-founded, mathematically sound foundation for learning models of sequential patterns from training data and for testing how well a particular model matches a sequence of observations.

In addition, an HMM framework has the advantage of allowing relations to be captured at the stroke level as well as at the object level within a hierarchical setup. This allows modeling contextual information about appearance of different types of ink (e.g., domain objects, free-form ink, text, editing operations), and the flow of the sketching process. For example, as shown in Fig. 5-3, when drawing a graphical representation of a linked list, people alternate between drawing domain objects (cells, pointers) and textual annotations (cell contents). Capturing higher level patterns of this sort can naturally be done within a hierarchical HMM framework. This would allow actions relevant to a particular task (e.g., writing vs. sketching) to be taken.

Next we briefly review HMMs, then we show how to combine the results from individual HMMs to perform recognition and segmentation for complex scenes with multiple objects.

5.2.2 Overview of HMMs

An HMM $\lambda(A, B, \pi)$ is a doubly stochastic process for producing a sequence of observed symbols. An HMM is specified by three parameters A, B, π. A is the transition probability matrix $a_{ij} = P(q_{t+1} = j | q_t = i)$, B is the observation probability distribution $B_j(v) = P(O_t = v | q_t = j)$, and π is the initial state distribution. $Q = \{q_1, q_2, ...q_N\}$ is the set of HMM states and $V = \{v_1, v_2, ...v_M\}$ is the set of observations symbols. Readers are referred to [43] for a comprehensive tutorial on HMMs.

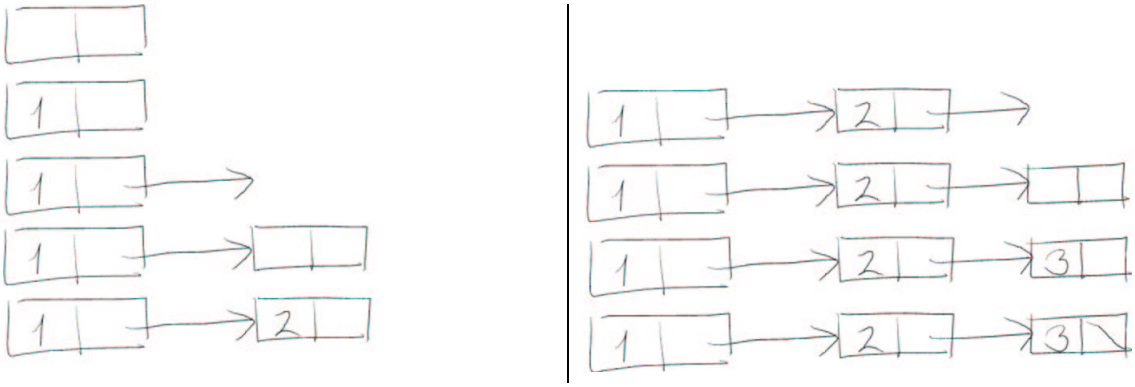

Figure 5-3: A typical way of drawing the linked list '(1 2 3).

In the toy example we discussed above, the observations v_i correspond to the encodings of input strokes in terms of lines of different orientations. The states Q can be thought of embedding the knowledge of having seen a particular observation sequence. The observation probability distribution $B_j(v)$ gives us the probability of observing each of the primitives given our current state. A captures the state transition dynamics.

Given an HMM $\lambda(A, B, \pi)$, we can answer three questions:

- Given a sequence of observations $O = o_1, o_2, .., o_k$, we can determine how well each model λ accounts for the observations. This is done efficiently in polynomial time using the Forward algorithm to compute $P(O|\lambda)$.

- For a given sequence of observations O, we can efficiently compute the best sequence of HMM state transitions for generating O. This is done using the Viterbi algorithm. We use this information in determining if a sequence of observations describe a complete object.

- Given a set of observations, we can estimate HMM parameters A, B and π to maximize $P(O|\lambda)$. This is done with the Baum-Welch parameter estimation procedure. Given training examples, we use this procedure to learn probabilistic models of how the user draws objects.

5.2.3 Modeling with HMMs

Encoding

As we alluded to in the example above, we encode sketches to generate a sequence of observations for recognition. We encode strokes using the Early Sketch Processing Toolkit described in [45]. This toolkit takes a raw stroke as input and returns a geometric approximation of it. The approximation can be an oval, line, polyline, curve, or a mixture of curves and polylines. We encode the output of the toolkit to convert sketches into discrete observation sequences. Our encoding has a total of 13 symbols. Four encode lines: positively sloped, negatively sloped, horizontal

and vertical. Three symbols encode ovals: circles, horizontal and vertical ovals. Four symbols encode polylines with 2, 3, 4, and 5+ edges, and one symbol encodes complex approximations (i.e., mixture of curves and lines). Finally we have a symbol that we use to denote two consecutive intersecting strokes. The choice of an encoding scheme is an important issue that can affect recognition accuracy. As we show, we obtained very promising results with this encoding.

As mentioned before, instances of the same object sketched in different styles may have encodings of different lengths. For example, if the user draws a square in four separate strokes instead of three, the corresponding encoding will be longer. Thus, our training data will have varying length encodings, and both training and classification routines have to be able to handle this. Below we propose two ways of formulating training and recognition using HMMs, one that uses a single model for each input length of each class, and another that trains one HMM for each class and uses variable length training data.

Modeling with fixed input length HMMs

Assume we have n object classes. Encodings of training data for class i may have varying lengths, so let $L_i = \{l_{i1}, l_{i2}, ...l_{ik}\}$ be the distinct encoding lengths for class i. We partition the training data into $K = \sum_{i=1}^{n} |L_i|$ sets such that each partition has training data for the same object with the same length. Now we train K HMMs, one for each set, using the Baum-Welch method. Note that each training set has uniform length, thus we know the input length for each HMM. In this framework, each class i is represented by $|L_i|$ HMMs, and we have an inverse mapping that tells us which HMM corresponds to each class.

With this setup, performing isolated object recognition is quite easy. We run the Forward procedure with the observation sequence O generated by encoding the isolated object. This procedure gives us the probability $P(O|\lambda_i)$. We do this for each HMM, find the model with the highest likelihood, and use the inverse mapping to get the object class. Unfortunately isolated object recognition requires the input sketch to be presegmented, which is usually not the case, and segmentation is itself a hard problem.

To simplify matters, for the moment assume that our recognition routines are called only when the sketching surface has complete objects. This is a strong assumption to make, and in section 5.2.3 we explain how it can be relaxed to do segmentation and recognition in presence of partially drawn objects.

Interpretation of a complex scene requires generating hypotheses for the whole scene. That is, it requires segmenting the entire observation sequence into subsequences and assigning models to these segments so that all the strokes in the scene are accounted for. This requirement of accounting for all strokes implies that individual interpretations should be chosen so that they form a globally coherent interpretation for the whole scene.

The hypothesis generation should be efficient, so combinatoric approaches are ruled out. The fact that individual HMMs return probability values makes it easy to define the objective for this stage, namely, choose interpretations that maximize the

probability corresponding to the entire scene. This is in fact an optimization problem that we solve using dynamic programming implemented in the form of a shortest path problem.

The shortest path in a graph $G(V, E)$ that we generate gives us the segmentation. We then perform classification as described above[1]. Segmentation and recognition begins by building the graph $G(V, E)$ such that: V consists of $|O|$ vertices, one per observation, and a special vertex v_f denoting the end of observations. Let k be the input length for model λ_i. Starting at the beginning of the observation O, for each observation symbol O_s, we take a substring $O_{s,s+k}$ and compute the loglikelihood of this substring given the current model, $log(P(O_{s,s+k}|\lambda_i))$. We then add a directed edge from vertex v_s to vertex v_{s+k} in the graph with an associated cost of $|log(P(O_{s,s+k}|\lambda_i))|$. No edges are added if the destination index $s + k$ exceeds the index of v_f. We complete the construction of G by repeating this operation for all models.

In the constructed graph, having a directed edge from vertex v_i to v_j with cost c means that it is possible to account for the observation sequence $O_{i,j}$ with some model with a loglikelihood of $-c$. The constructed graph may have multiple edges connecting two vertices, each with different costs. By computing the shortest path from v_1 to v_f in G, we minimize a sum of negative loglikelihoods, equivalent to maximizing the likelihood of the observation O. The indices of the shortest path give us the segmentation. Classification is achieved by finding the models that account for each computed segment.

A nice feature of the graph-based approach is that while the shortest path in G gives us the most likely segmentation of the input, it is also possible to get the next k-best segmentations using a k-shortest path algorithm. We use the algorithm described in [15] to get alternate segmentations. This information can be used by another algorithm for dealing with ambiguities or by the user, as done in speech recognition systems with n-best lists.

Modeling using HMMs with variable length training data

The formulation above makes the construction of the graph G easy because each HMM is trained using fixed length data. At each step s, we can easily compute the destination of the edge originating from the current vertex, v_s, by adding the input length for λ_i to s. However, one drawback of the above method is that it requires an artificial partitioning of the training data for each model, dictated by the variations in description lengths for the same object. Such an artificial partitioning reduces the total number of training examples per model and prevents representing similar parts of different sketching styles with the same HMM graph fragment, which reduces recognition accuracy and increases cumulative model sizes.

We avoid the artificial partitioning of the training data by grouping the data for all sketching styles together, and training one HMM per object class. After the training

[1]The graph $G(V, E)$ that we build for segmentation is distinct from the graphs that represent HMM topologies.

is over, for each model we also estimate the probability of ending at each state q of λ_i by getting the ending states for the training examples using the corresponding Viterbi paths. This information is used during recognition.

The graph G has the same number of nodes as the previous approach. We generate it by iterating over each model λ_i, adding edges with the following steps: for each observation symbol O_s, we take a substring $O_{s,s+k}$ for each $k \in L_i$. Next we compute the loglikelihood for the observation given the current model, $log(P(O_{s,s+k}|\lambda_i))$, and add a directed edge from vertex v_s to vertex v_{s+k} in the graph with an associated cost of $|log(P(O_{s,s+k}|\lambda_i))|$.

We augment each weight in the graph with a term that accounts for the probability that $O_{s,s+k}$ is the encoding of a complete object (i.e., the probability of $O_{s,s+k}$ leaving λ_i in one of its terminal states). This is required because if a model λ can emit the sequence O with high probability, then it can also emit any prefix O' with high probability (because it must emit O' before it can emit O). As a result, if $P(O|\lambda)$ has high probability, then for a prefix O' of O, $P(O'|\lambda)$ will have high probability. This means that if we have two ways of sketching an object, one using m strokes and the other n, $m < n$, then every time we draw the object using the n stroke style, the edge $\langle v_s, v_{s+n} \rangle$ corresponding to the complete object will be added to G, but the edge $\langle v_s, v_{s+m} \rangle$ corresponding to the first few strokes of the object will also be added. Ideally in a case like this where a complete object is drawn, we would like to use the edge corresponding to the complete object. This is achieved by penalizing edges corresponding to incomplete objects, by testing whether the observation used for that edge puts λ_i in one of its final states using the ending probabilities estimated earlier. Segmentation and recognition is achieved by computing the shortest path in G as described above.

Handling partial drawings

Earlier, we assumed our recognition routine was called only after the user completed each object. For example, if the user previously drew a number of objects, and put down two of the four strokes making up a square, the recognition routine is not run. It is only after the user completes the square that we start hypothesis generation. In most setups, this is a very strong assumption, because it requires explicit input from the user or another external system indicating that the scene is complete.

Instead, the recognition system should be able to update its hypotheses after each stroke is added, whether or not this stroke completes the current object. This means we need a way of interpreting a scene reliably even if its encoding ends with observations corresponding to a partially drawn object. If the observations from partially drawn objects are not modeled explicitly, the global coherence condition, which states that all strokes should be accounted for, results in favoring many incorrect interpretations for the sake of accounting the for observations coming from the partially drawn object.

Given an observation O describing a scene with a partially drawn object, ideally we would like to identify strokes that seem to be the beginning of an object and tell what object it is most likely to be. There is an elegant way of achieving this within

the framework we described. As mentioned before, if $P(O|\lambda)$ has high probability, then for any prefix O' of O, $P(O'|\lambda)$ has high probability. We take advantage of this property to recognize partial objects appearing at the end of a stroke stream. In the graph construction stage, if the destination index $s+k$ exceeds the index of v_f, then instead of trying to link v_s to v_{s+k}, we put a directed edge from v_s to the final node v_f. We set the weight of the edge to $|log(P(O_{s,|O|}|\lambda_i))|$. Here $O_{s,|O|}$ is the suffix of O starting at index s. The segmentation algorithms described above work without modification.

5.3 Proposal for an improved recognition method

We wrote an early implementation of the above ideas mainly for proof of concept purposes. The recognition results we got were promising. This early implementation also raised interesting research questions and revealed some weaknesses of the approach as listed below:

- Effects of the encoding scheme: The discrimination power of the statistical model depends on the nature of the employed encoding scheme. The encoding scheme described above is fairly simple, and in some cases it generates inadequate observation sequences. As we describe in detail later in this section, a good encoding scheme should address the spatial, two-dimensional nature of sketches.

- Lack of structural constraint checking: HMMs have been used in the literature very successfully to model time series. On the other hand, the kinds of sketches we would like to recognize not only have a spatial, two-dimensional component, they also lend themselves well to structural recognition methods. A tight integration with structural recognition routines has computational and robustness advantages.

5.3.1 Effects of the encoding scheme

There is no magic recipe for choosing a good encoding scheme for an HMM based recognition architecture. The general principle is to have an encoding that captures the underlying Markovian process. Although there are methods for determining subsets of important features from a given feature pool, the choice of encoding is usually based on the domain knowledge about the problem at hand. In our case, we would like our encoding to be translation, rotation, and scale invariant. In addition, we want objects from different classes to generate different observation sequences. For example, using the encoding in the previous section, we get the same encoding for both objects shown in Fig. 5-4.

The encoding scheme we propose is to describe primitives by their relative orientation, length, and position. Given a sequence of primitives[2] $l_1, l_2, ...l_n$, we generate

[2]For simplicity, assume that lines are the only the primitives we have.

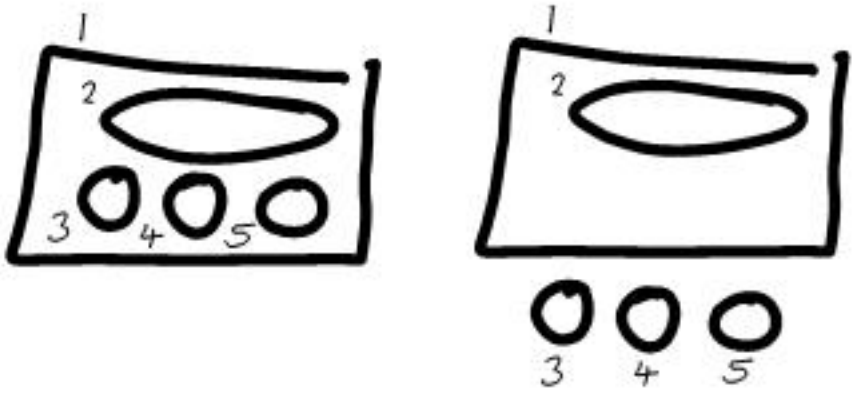

Figure 5-4: Two sketches that generate the same observations sequences using our initial encoding. Numbers next to each stroke indicate the drawing order.

observations $O_1, O_2, ...O_n$ such that:

$$O_i = \begin{bmatrix} l_{i+1}.length/l_i.length \\ t_i.x/l_i.length \\ t_i.y/l_i.length \\ \Delta\theta \end{bmatrix}$$

where t_i is the vector corresponding to the translation from the end of l_i to the midpoint of l_{i+1} measured in a coordinate frame centered at the first point of l_i such that the end point of l_i has the coordinates $(l_i.length, 0)$. The $\Delta\theta$ term captures the relative orientations of the primitives.

5.4 Using structural constraints

As we have established in our user study, stroke ordering follows a consistent pattern for most people. Yet the kinds of objects we are interested in have spatial, geometric and topological properties that are not sequential or one dimensional. These properties can be described in terms of constraints that hold between components of objects, and recognition can be achieved by a class of methods, generally referred to as structural approaches to recognition, that have been well studied and employed successfully in syntactic recognition [21], visual languages [36], and model based vision communities [42]. The recognition scheme that we described in chapter 4 falls under this category.

We believe that exploiting the sequential, incremental nature of sketching using HMMs, and the geometric, spatial, topological regularities by structural methods is a promising research direction to explore. Instead of treating the two approaches as separate black boxes working in isolation, we propose a unified framework where these methods can share information about their hypotheses. We believe this will

reduce error rates and increase efficiency.

Embedding structural information in the HMMs

HMMs are suitable for tasks where the Markov assumption holds. The first order Markov assumption states that the next state is dependent only on the current state. On the other hand, if we wish the state of our models to depend on spatial and geometric relationships between the latest primitive and multiple past primitives (as opposed to just the previous one), then the Markov assumption is violated. Although it is possible to build Markov models of higher order where the current state depends on the previous n states, this quickly inflates the conditional probability tables specifying the system dynamics, especially if we want to capture primitive relationships for objects with a large number of primitives.

Another possibility is to use two dimensional Hidden Markov Models introduced by Najmi and Gray [31]. The drawback of this approach is that it assumes a lattice neighborhood structure that is well suited for images with pixel neighborhoods on an mxn lattice, but not for sketches where there is no natural neighborhood structure to the primitives.

Rather than trying to resolve structural relationships using HMM variants, we propose to attach structural meaning to the states of our model. In particular, we would like the states of our model to correspond to partial interpretations of the object in question. This mapping is learned after the parameters of the HMM are estimated. Unlike the initial model described in sections 5.2.3 and 5.2.3, we need labeled training data to derive the mapping between states and the partial interpretations[3]. To derive the mapping, for each training example $l_1, l_2, ...l_n$, we derive the observation sequence $o_1, o_2, ...o_m$ and find the corresponding most likely state sequence $s_1, s_2, ...s_m$. We also derive a sequence of partial interpretations $I_1, I_2, ...I_m$. For each state s_i, we maintain a lookup table to store the mapping between states and partial interpretations. The mapping would be a probability distribution over likely partial interpretations for that state.

Attaching such meaning to the states will allow us to get an estimate of the possible partial interpretations, which can be accomplished by deriving the most likely state sequence for the given observations and then do a table look up to determine the most likely partial interpretation sequence. Such a sequence of most likely interpretations builds the bridge between the Markov model and the structural matching approach enabling the following:

- Structural constraints of the generated hypotheses can be tested, and the results can further enforce or weaken the belief in those interpretations. As we mentioned earlier, the HMM framework allows us to derive an n-best list of interpretations. Just as in speech recognition, the correct interpretation of the observations may not be the first in this list. Geometric and spatial properties of the most likely interpretations in the n-best list can be checked against the structural model, pruning unplausible interpretations.

[3]The training data must have class and stroke level correspondence information.

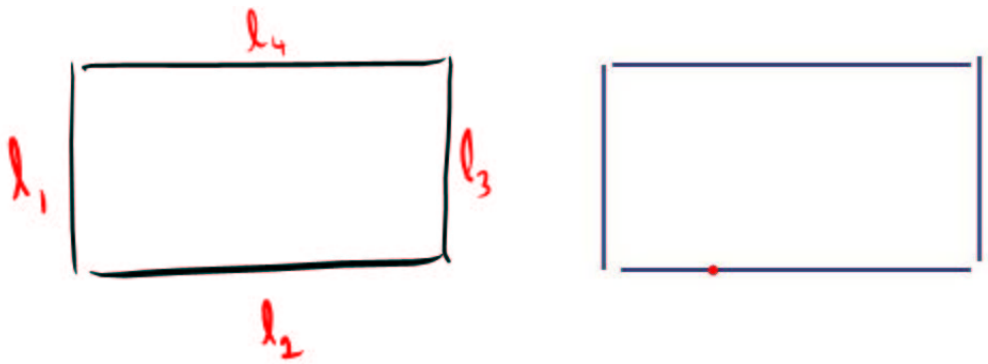

Figure 5-5: Example of an over-segmented primitive. Instead of being interpreted as a single line, l_2 is incorrectly broken into two lines by the lower level processing routines.

- Suppose the user starts drawing an object in a novel drawing order, but finishes the rest in an ordering that the model knows about. Then the structural matching can be used to derive partial interpretations for the portion of the sketch with novel stroke ordering, and the HMM can pick up recognition from the state corresponding to the partial interpretation where the user switches to the known drawing order.

- Additional information about possible labelings of primitives can be attached to the lookup tables at each state, giving good initial guesses on most likely correspondences. This can be thought of as a form of indexing where the index is computed by finding the most likely HMM state for an observation sequence, and the indexed information is the possible mappings between recently seen strokes and model features.

5.4.1 Robustness

As with most encoding schemes, the new encoding scheme we presented is susceptible to noise in the sketch, (i.e., over-segmented or spurious primitives.). We propose to address the over-segmentation problem (see Fig. 5-5) by modifying the forward and the Viterbi algorithms so they can accept multiple observation sequences generated from a single sketch and choose the one that matches the models with the highest likelihood.

With the encoding scheme above, if a primitive is over-segmented, this results in a new observation sequence. For example, if $O = \{O_1, O_2, ..O_i, O_{i+1}, O_{i+2}, O_{i+3}, ..O_n\}$ is the correct observation sequence (i.e., without over-segmentation), and primitive l_i is incorrectly over-segmented by the lower level stroke processing routines, then we will have a new observation sequence one observation longer in length. The new sequence will be the same as the original one except a three unit long substring starting after the i^{th} observation. The new observation sequence will be: $O = \left\{O_1, O_2, ..O_i, O'_{i+1}, O'_{i+2}, O'_{i+3}, O_{i+3}, ..O_n\right\}$ O and O' are the same except this

substring. We will modify the forward probability calculation and the Viterbi path algorithm such that the observation sequence that better matches the model is found efficiently (i.e., without worsening the time complexity of these algorithms).

Chapter 6

Proposed evaluation

6.1 Data collection

Evaluation of our system will require collecting data for testing and training. In the data collection process, we will focus on at least two of course of action diagrams, electrical engineering drawings, UML diagrams, and mechanical engineering drawings. The course of action diagrams domain is of particular interest to us because it contains many objects with rich variations (Fig. 6-1).

Subjects contributing data will be instructed to draw as they would do normally. An important issue to consider is if the users should be informed about the kinds of sketches that our system can handle. For example, our system cannot handle artistic renderings, over-tracing and shading. One option is to make sure users are aware of these limitations. Another option is to let the users sketch as they wish and eliminate drawings with features that we don't support. We will decide on which approach to take by running pilot data collection sessions.

6.2 Evaluation of the model-based recognition approach

The primary contribution of our model-based recognition approach is a more informed exploration of the search space. In order to test how informed our search is, we will compare the performance of our algorithm to a baseline algorithm (e.g., the modified interpretation tree algorithm described in section 4.2.1).

The success of the system will be measured by the degree to which it remains conservative in creating partial interpretations. We will measure this by comparing the number of partial interpretations generated by both methods during recognition of objects chosen from the collected test data. Percentage decrease in the generated partial interpretations will be the relevant metric.

We would also like to measure how recognizers perform when presented with objects from different classes. Specifically, we are interested in determining whether the number of partial interpretations generated by our approach is still relatively

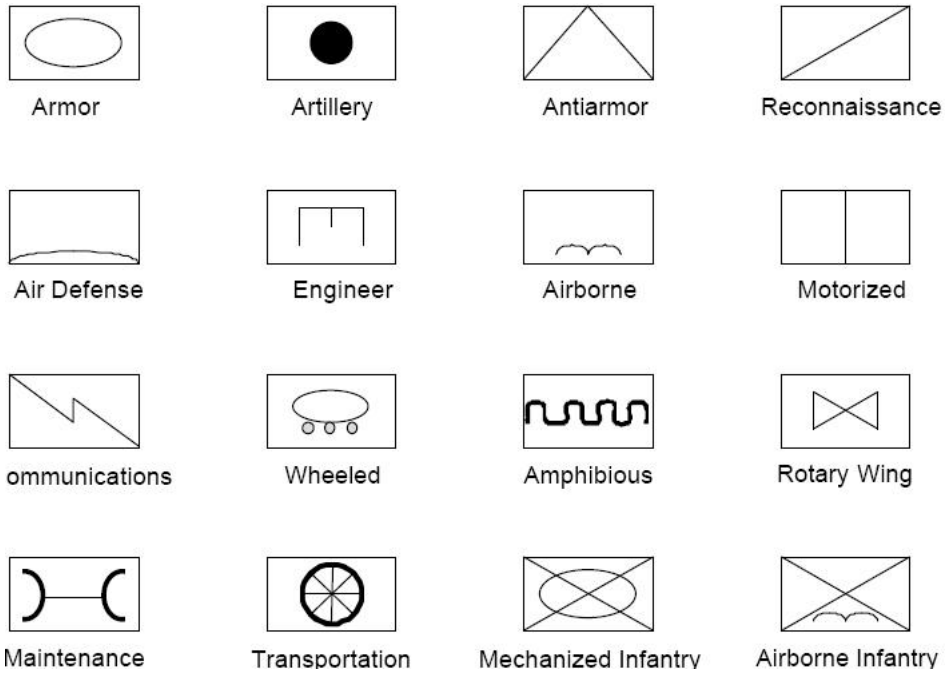

Figure 6-1: Symbols used in course of action diagrams.

low compared to the baseline method for each recognizer, when the scene contains objects from other object classes. Generating few partial interpretations for scenes with objects from other classes is an important feature to have in recognizers, because during sketch recognition, all recognizers will run in parallel while only one recognizer will be relevant. In this case, we would like the irrelevant recognizers to generate as few, preferably no, partial interpretations for the scene. Again, we will count the number of false positives for our method and the baseline method, and compare the percentages.

6.3 Evaluation of the HMM based method

In order to evaluate the HMM based recognition method, we will train models using data collected from users and focus our tests on the recognition rates as well as on how the system handles drawing styles that it hasn't been trained on.

6.3.1 Testing

False positive and false negative ratios

The data collected from each user will be partitioned into training and test sets. For each test set, we will measure the percentage of false positives and false negatives for each data set.

System speed

We will also measure the system's speed in terms of average time spent recognizing each object. Because objects may have different number of primitives, a better measure of speed could be the average time spent per object primitive.

Effects of the encoding scheme

We would like to measure the effects of using different encoding schemes on the recognition. In particular, we would like to see how using pairs of strokes for encoding input compares to using the simpler encoding scheme described in section 5.2.3.

Another issue is the amount of training data that will be needed for parameter estimation without overfitting for different encoding schemes. Depending on the complexity of the encoding scheme, the amount of data needed for parameter estimation will vary. One would expect the simpler encoding scheme to generalize from fewer examples, although its discriminative power will be poor because it doesn't capture relative position, orientation and scale. On the other hand the more elaborate encoding scheme will have better discriminative power, but will require more training examples for reliable paramater estimation.

Failure behavior

An important question to ask is, what happens when the system comes across a series of strokes it can't recognize. This may happen because of spurious strokes that do not belong to any object, or because the system came across a novel sketching style. In this case, we would like to know if the effects of the failure on the overall recognition will be graceful or catastrophic.

Failures will be triggered by using sketches with spurious strokes and novel objects. Ideally we would like the effects of any misrecognition to remain relatively local, and not cause objects sketched far away in time to be misrecognized. We will measure the magnitude of failure caused by spurious strokes and novel objects by compiling statistics on the size of the misrecognition neighborhood, defined as the number of misclassified objects before or after the source of error.

Handling over-segmented features

An important feature of our approach is its robustness in presence of low-level over-segmentation errors. An example of an over-segmented primitive from chapter 5 is reproduced in Fig. 6-2: Line l_2 is incorrectly broken into two lines by the lower level processing routines.

We will compile a test set of objects with over-segmented primitives obtained from two sources. The first source will be the sketches collected in the data collection for evaluation as explained in 6.1. We will also produce synthetic test data by explicitly adding over-segmented primitives by simulating errors in the low level processing. Our evaluation will measure the percentage of correctly recognized objects in presence of over-segmented primitives.

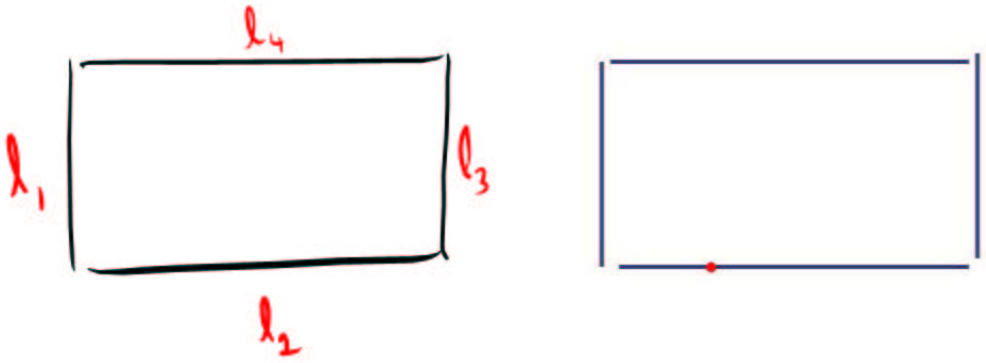

Figure 6-2: Example of an over-segmented primitive. Instead of being interpreted as a single line, l_2 is incorrectly broken into two lines by the lower level processing routines. Our evaluation will test the behavior of our methods in such scenarios.

Bibliography

[1] GREC 99, LNCS volume 1941 Springer, 2000.

[2] Aaron Adler. Segmentation and alignment of speech and sketching in a design environment. *Masters Thesis, Cambridge, MIT*, February 2003.

[3] H. A. Almohamad and S. O. Duffuaa. A linear programming approach for the weighted graph matching problem. *IEEE pattern analysis and machine intelligence*, Vol 15, No 5, 522-525, 1993.

[4] Christine Alvarado, Mike Oltmans, and Randall Davis. A framework for multi-domain sketch recognition. *AAAI Spring Symposium: Sketch Understanding*, 2002.

[5] Farshid Arman and J. K. Aggarwal. Cad-based vision: Object recognition strategies in cluttered range images using recognition strategies. *CVGIP: Image Understanding*, Vol 58, No 1, 33-48, 1993.

[6] Song Baohua, Lu Changde, and Yang Haicheng. Research on product-tree based multimode and integrated product model. *In Proceeding of the 6th International Conference On Manufacturing Technology, Hong Kong.*, Dec 16-18, 2001.

[7] J. Byne and J. Anderson. A cad-based computer vision system. *Image and Computing Vision*, Vol 16, 533-539, 1998.

[8] Andrea Califano and Rakesh Mohan. Multidimensional indexing for recognizing visual shapes. *IEEE Transaction on Pattern Analysis And Machine Intelligence*, Vol. 16, No. 4, pp 373-391, 1994.

[9] Todd Cass. Polynomial-time geometric matching for object recognition. *International Journal of Computer Vision*, 21(1/2),37-61 (1997).

[10] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. *3rd IAPR-TC15 Workshop on Graph-based Representation*, 2001.

[11] Mauro Costa and Linda G. Shapiro. Object recognition and pose with relational indexing. *Computer Vision and Image Understanding*, v79, 364-477, 2000.

[12] Fred DePiero and David Krout. Lerp: An algorithm using length-r paths to determine subgraph isomorphism. *Pattern Recognition Letters*, Vol. 24, pp 33-46, 2003.

[13] Jacob Eisenstein. *placeholder entry.*

[14] D. Eppstein, Z. Galil, and G. Italiano. Dynamic graph algorithms. *In CRC Handbook of Algorithms and Theory of Computation*, Chapter 22. 1997.

[15] David Eppstein. Finding the k shortest paths. *SIAM J. Computing Vol.28, No.2, pp. 652-673*, February 1988.

[16] David Eppstein. Subgraph isomorphism in planar graphs and related problems. *Journal of graph algorithms and applications*, Vol 3, No 3, 1-27, 1999.

[17] R. Fisher. Non-wildcard matching beats the interpretation tree. *In Proc. British Machine Vision Conference (BMVC92)*, pages 560–569, Leeds, UK, September 1992.

[18] Robert B. Fisher. Performance comparison of ten variations on the interpretation-tree matching algorithm. In *ECCV (1)*, pages 507–512, 1994.

[19] Ken Forbus. placeholder entry.

[20] Andrew S. Forsberg, Mark Dieterich, and Robert C. Zeleznik. The music notepad. In *ACM Symposium on User Interface Software and Technology*, pages 203–210, 1998.

[21] K. S. Fu. Syntactic pattern recognition and applications. *Prentice-Hall, Englewood Cliffs, NJ*, 1982.

[22] Chris Goad. Special purpose automatic programming for 3d model based vison. *DARPA IUW*, pp 94-104, 1983.

[23] W. Eric L. Grimson. The combinatorics of heuristic search termination for object recognition in cluttered environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), Volume 13, 929-935*, September 1991.

[24] M. Gross. Recognizing and interpreting diagrams in design. In *2nd Annual International Conference on Image Processing*, pages 308–311, 1995.

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

[26] Tracy Hammond. A domain description language for sketch recognition. *Proceedings of 2002 SOW*, 2002.

[27] Charles Hansen and Thomas C. Henderson. Cagd-based computer vision. *IEEE Transactions on pattern analysis and machine intelligence*, Vol 11, No 11, 1989.

[28] Katsushi Ikeuchi and Takeo Kanade. Automatic generation of object recognition programs. *Proceedings of the IEEE*, Vol 76, No 8, 1988.

[29] Kunihiro Kondou, Naoki Kato, and Toshinori Watanabe. Osr:online sketch recognition by data compression technique. *IPSJ JOURNAL Abstract Vol.38 No.12 - 007 pp. 2468-78*, 1997.

[30] 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.

[31] Jia Li, Robert M. Gray, and Richard A. Olshen. Multiresolution image classification by hierarchical modeling with two dimensional hidden markov models. *Transactions on Information Theory, 46(5):1826-41*, August 2000.

[32] James V. Mahoney and Markus P. J. Fromherz. Handling ambiguity in constraint-based recognition of stick figure sketches.

[33] James V. Mahoney and Markus P. J. Fromherz. Interpreting sloppy stick figures by graph rectification and constraint-based matching. *Fourth IAPR Int. Workshop on Graphics Recognition, Kingston, Ontario, Canada.*

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

[35] Csar Pimentel Manuel J. Fonseca and Joaquim A. Jorge. Cali: An online scribble recognizer for calligraphic interfaces. *AAAI Spring Symposium: Sketch Understanding, March 25-27, Stanford CA*, 2002.

[36] K. Marriot and B. Meyer. A survey of visual language specification and recognition. *Theory of Visual Languages. Springer Verlag*, June 1998.

[37] B. Messmer and H. Bunke. Fast error-correcting graph isomorphism based on model precompilation. *Technischer Bericht IAM-96-012, Institut fur Informatik*, 1996.

[38] Bruno T. Messmer and H. Bunke. Subgraph isomorphism in polynomial time. Technical Report IAM 95-003, 1995.

[39] Clark F. Olson. Probabilistic indexing for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(5),518-522 (1995).

[40] MIT people.

[41] OGI people.

[42] Arthur Pope. Model-based object recognition: A survey of recent literature. *TR 94-04*, January 1994.

[43] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE, Vol. 77, no 2.*, February 1989.

[44] Dean Rubine. Specifying gestures by example. *Computer Graphics*, 25(4):329–337, 1991.

[45] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. *Proceedings of PUI-2001*, November 2001.

[46] Michael Shilman, Hanna Pasula, Stuart Russel, and Richard Newton. Statistical visual language models for ink parsing. *AAAI Spring Symposium: Sketch Understanding, March 25-27, Stanford CA*, 2002.

[47] F. Stein and G. Medioni. Structural indexing: Efficient 3-d object recognition. *IEEE Transaction on Pattern Analysis And Machine Intelligence*, Vol. 12, No. 2, pp 125-125, 1992.

[48] Edward Tsang. Foundations of constraint satisfaction. *Academic Press, London and San Diego*, 1993.

[49] David G. Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers and Graphics*, 14(2):263–274, 1990.

[50] Shinji Umeyama. An eigendecomposition approach to weighted graph matching problems. *IEEE pattern analysis and machine intelligence*, Vol 10, No 5, 695-703, 1988.

[51] Olya Veselova. Perceptually based learning of shape descriptions. *Masters Thesis, Cambridge, MIT*, 2003.

[52] Toshinori Watanabe, Ken Sugawara, and Hiroshi Sugihara. A new pattern representation scheme using data compression. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24(5): 579-590*, 2002.

[53] Liu Wenyin. Example-driven graphics recognition. *Structural, Syntactic, and Statistical Pattern Recognition, LNCS 2396*, 2002.