DIAGRAM RECOGNITION USING HIDDEN MARKOV MODELS

David Henry & Aster Wardhani

Centre for IT Innovation
Faculty of Information Technology
Queensland University of Technology
2 George St
Brisbane, QLD
Email: {a.wardhani@, dw.henry@student.}qut.edu.au

ABSTRACT

User input interfaces are quickly become more natural and intuative, relying less and less on the traditional mouse and keyboard interface, and moving towards a pen based input system. Current technologies which take advancement of these developments in hardware have been based primarily on handwriting recognition to allow the user to write there instructions instead of typing. Some work has been performed regarding diagram recognition from on-line input, however these system have been developed using hardcoded parameters with fuzzy sets and common feature sets.

Hidden Markov Models are a powerful mathematical recognition tool, which is current being used in feilds such as speech recognition, handwriting recognition and DNA cell searching and classification applications. This is the first attempt at using a hidden markov model to recognize input from a two dimensonal spacial environment, the result of the initial implementation have shown promising results for more complex recognition models.

1. INTRODUCTION

The aim of this project is to develop a framework for input, processing and recognition of hand drawn diagrams using Hidden Markov Model (HMM). Current technologies supporting pen based input devices include: Digital Pen by Logitech [1], SmartPad by Pocket PC [2] and Digital Tablet's by Wacom [3] and Tablet PCs by Compaq [4]. The emergence of pen-based user interactive tool has increased the need for more hand-based recognition techniques. Currently, this tool has been used primarily for direct input or used as a substitute mouse interface. There have been numerous works on handwriting recognition [5], however works such as diagram recognition is still limited. Using pen device, engineers in the field, or online collaborators, etc, can sketch various types of diagrams and rather than just store them as bitmap, they can be converted into vector formats using diagram recognition. This can then be imported into various applications.

Existing work in diagram recognition include: on-line Scribble Recognizer [6] and Distributed Architectures for Pen-Based Input and Diagram Recognition [7]. The scribble recognizer uses object identification based on a finished image, without using real-time input streams. It describes common feature of varies objects and how statistics can use to create a fuzzy set which can then be used to classify each object. The approach is shown to be fast and reasonably flexible based on the features used to identify each diagram object. While this approach shows promising results from the trials conducted, it does not provide any real-time processing of the users input streams.

The pen-based architecture provides the implementation of a pen-based input system using a PDA front-end and desktop computer back-end. The PDA is responsible for low level object recognition and editing, while high-level recognition is performed by the desktop PC. The use of the PDA input device allows diagram input in the field which can be uploaded to the desktop PC on return to the office or real time communication between the PDA and PC if used in the office.

This provides an initial approach for real-time processing. The main aim of the system implementation being the development of standard system architecture, applicable to any diagram recognition application requiring high performance and / or mobility. They proposed the use of layered design architecture. The lowest level is the domain invariant system to input stream from the pen, include x-y coordinates and directional/velocity vectors. Input streams are combined in strokes, in the next layer, this is performed using fuzzy sets to determine the the stroke type based on preset criteria. Strokes are then combined to form basic shapes such as polygons, lines and circles. The top level of the design is responsible for grouping shapes into complex diagrams. The limitations to the system include slow recog-

nition, small screen size and slow communication between PC and PDA. The above system limitations are a result of inadequate hardware for the processing task, with resent advances in technology it is expected that these will be short term problems.

Existing technologies such as voice and handwriting recognition have successfully shown that HMM can be implemented within a design framework to perform statistically based recognition tasks. HMM's can be used in systems to recognize and predict outcomes using only an indirect knowledge of the users input sequence. Speech recognition is a series of unique sounds while handwriting recognition is a series of unique stokes patterns; these patterns repeated in different orders are used to determine the most probable users input. This is done by calculating the statistical probability of possible inputs, at a word level using a trained model, and at a symbol level using the actual input. A number of papers have investigated the use of HMM's for cursive handwriting recognition however no research is currently published relating to diagram recognition using HMM's. One report [5] used a HMM speech recognition model as the basis for a cursive handwriting recognition application. This resulted in error rates as low as 1.1% when using both context and grammar to determine individual words and sentence structure.

The HMM design is proposed as a suitable recognizer for diagram recognition due to its ability to recognize complex object representations without the need for a complete understanding of all system parameters. Added to this the HMM is a flexible and increasingly popular recognition tools which has shown its ability to revolutionalise recognition tasks for speech and handwriting applications amongst others. The HMM implementation can be designed as a separate module within the system and hence can be easily modified and/or updated to include new shapes, design objects. The modularity also always different diagram formats to be processed by simply changing the HMM module being used.

Another advantage of HMM is their ability to be recalibrated using tested training algorithms. This allows the recognition process itself to be modified to the user input patterns, while other methods currently being investigated rely on more complex fuzzy sets or common feature sets which need to be reprogrammed for each new shape which is identified within the system.

2. PROPOSED SYSTEM

A major difference between diagram recognition and other applications using HMM is the non-linearity of the input stream. Both speech and handwriting recognition applications are intrinsically linear. When writing, standard convention dictates that the input is entered from left to right horizontally on the page with the sentence structure following a logical and preset format. A diagram however, does not have the same constraints, with hundreds of different ways that even the simplest shape can be drawn. Added to this there is no requirement that a shape be completed before starting the next shape within the drawing as there is in handwriting. To overcome this non-linear input of symbols, the data must be preprocessed once the input stage is finished. This processed data can then be passed to the HMM for recognition. This results in system which is not capable of real-time recognition.

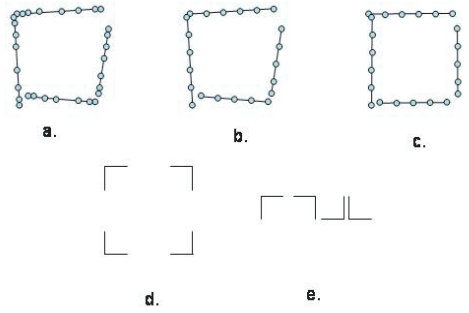

Fig. 1. Symbol Generation Phases - a. Point input stream. b. Normalized point stream. c. Line type and rotation determined. d. Symbol representation. e. Ordered symbol list.

The system is implemented and tested using the Wacom Graphire tablet. This device uses a USB interface to the computer. However the same principple can be applied to other pen-based sevices.

The proposed system will process the user input pointstream, however, no real-time processing has been implemented at this stage, refer to Section 4.

2.1. Data input

The input stream is captured using the MFC messaging system. The advantage of this initial implementation is that all Microsoft supported devises can now be used within the application.

The input stream is generated at regular time intervals, this input stream is captured whenever the primary button is depressed. This results in a data stream where the points are spaced inversely proportional to the speed of mouse movement at the time of input, refer Figure 1-a.

2.2. Data Normalization

During the input process the captured data stream is normalized to a point stream where each point is a fixed distance apart. This processing is performed to assist in minimizing errors during data processing. Figure 2-b illustrates the data stream from the input process of a straight line drawn within

Fig. 2. Each circle represents the data stream as seen by the system. a) Shows the stream which is generated by the system messages, note the variable intervals, this is produced by moving the mouse at different speeds as the points are generated at regular time intervals. b) Shows the data stream after normalization has occurred.

the system drawing window, while Figure 1-b shows the output of the normalization process. To ensure the normalization process does not alter the user inputed data stream the length of the finial point in the list will be adjusted so that it falls on the last point of the user input.
Data Normalization is performed using the following algorithm:

```
While data input exists.
  Measure distance to next input point, d (1).
  Calculate distance until next
   normalized point, 1 (2).

  If l < 0
    While l < 0
      Create new Normalized Point
      l = l + normalized distance
  Else
    process next point
```

Equation 1 - $d = \sqrt{x_{diff}^2 + y_{diff}^2}$
Equation 2 - $l = l - d$

2.3. Corner detection and line segmentation

Once the data has been normalized it is segmented into line intervals. This segmentation occurs when either a corner is detected by the system or the input stream is stopped (i.e. input device leaves input area or the primary "'draw"' button is released).

Corner detection is performed by simply comparing the line angles of adjoining normalized points. If the difference between the adjoining segments is larger than a present value then a corner point is generated and a new input stream is created.

This method of line segmentation produces roughly straight line segments, and allows the drawing style of the user to be ignored in terms of implemented a predefined input order or method. This is achieved by breaking the user input up into straight line segments, as a result a shape which is drawn in a single pen movement is equivalent to the same shape

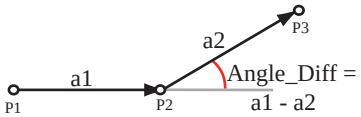

Fig. 3. Three normalized points (P1, P2 and P3) are used to calculate two adjoining line angles (a1 and a2). The difference between these two line angles determines the Angle Difference (Angle_Diff). If the value of Angle_Diff is greater than the system parameter maxCornerAngle, then a corner is detected.

being drawn with $\hat{x}$ ($\hat{x}$ = any integer number of elements greater than 1) separate pen movements.

2.4. Connect & Classify Line Types

The system is also able to recognize a single line segment if it is drawn in one movement or in $\hat{y}$ ($\hat{y}$ = any integer number of elements greater than 1) short sections. This recognition is achieved by attempting to join all line segments end-to-end with all other line segments, updating the list where the ends meet. If more than two line segments meet at a single point then none of these line segments can be joined together (can still be joined at the other end if line segments meet. This is due to the current systems limitation of only processing line segments at line ends in later stages of the system, refer section 4.

Once the line interval data has finished processing it is analyzed to determine its type. There are two recognized types of line segments, straight line and arc. This classification is performed based on:

- Average distance between input stream and a line joining the start and stop points of the line segment.
- The ratio of the distance between the start and stop points, to the maximum distance along a perpendicular line to the user inputted data point

When a straight line segment is selected it is classified into a rotational angle, the current system implements 4 possible angle selections, each represents two of the cardinal points, Figure 1-c shows how a set of input points are normalized to the possible rotational values.

To eliminate unnecessary processing cycles all line segments which consist of less than 3 normalized points are ignored for the processing stage.

At this stage arc segments are ignored by the system, refer 4.

2.5. Generate Symbols

DEFINITION

Symbol - In HMM theory symbols refer to a simple repeated data pattern which can be easily identified within the data set. Symbols are generated from the users input, the set of symbols should be capable of adequately representing all parts of the user input. Symbol selection is an important part of ensuring HMM's successfully recognize user input into the system.

The symbol generation process is the key to abstracting the users input sequence from the recognition model. Each symbol is representative of a change point in the diagram, as it is a joining point on small line segments. Currently only using Straight line in symbol generation, refer to Section 4.

Each symbol that is generated by the system should be representative of the users input. At present only two distinct user generated symbols are recognized, they are a 90^o angle and a 45^o angle where the second arm - anti-clockwise rotation - is at least 2.5 time longer that the first arm.

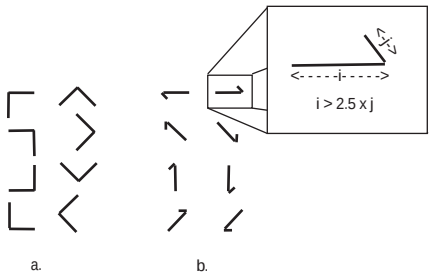

Fig. 4. Current user input generated symbols - a. 90^o angles. b. 45^o angles.

The symbol generation process works by performing a linear search on all sets of straight line elements. If a symbol element is detected then a factory method is used to generate the corresponding symbol. This allows the system to easily add new symbol definitions.

Once a symbol has been detected its rotation must also be calculated. At present the system supports eight different rotation angles, each 45^o from the next, refer Section 4.

In addition to the symbols generated from the user input there are also a number of system generated control symbols. These symbols are used to assist in the HMM recognition process. The following system generated symbols are currently recognized by the system:

1. Space - Used to separate object sequences from each other. This symbol indicates that the users would normally start a new diagram object at this point in time.
2. Apostrophe - Used within an object when a line splits into two elements.
3. Start - Used to indicate the start of an input sequence.
4. Stop - Used to indicate the end of an input sequence.

These system generation symbols will be discussed in more detail in Section 2.7.

2.6. Generate symbol lists for shape and join objects

Once all possible symbols have been generated it is a matter to join these symbols together to form ordered lists. This process has two steps; firstly join all possible symbols together, then determine the type of list it forms and order the symbol list.

Symbol lists are generated recursively using the following pseudo algorithm:

```
While Symbols remain in list
  Select the first symbol, S1, in the
   symbol list.
  Delete symbol, S1, from the symbol list.
  Loop while shape not detected.
    Call search procedure,
     returning list of joined symbols, PJoinL.
    If PJoinL has more than one symbol.
      Search all possible links from all
       symbols, created shape if possible.
      If no joined shape can be created.
        Ignore shape as too complex.
      Else.
        Add the Apostrophe symbol to
         indicate where the shape splits.
        Add as Symbol List object.
        Shape Detected so start
         processing again.
    Else if PJoinL is empty.
      Invert list.
      Attempt to process from new end.
      Exit Procedure.
    Else.
      Update S1, to next symbol in PJoinL.
      If S1 joins to first symbol in list.
        Create Shape List object, SList.
        Shape Detected process next symbol.

Search procedure
-----------------
Select the first line element, L1, from symbol.
For all Symbols, Si, in the symbol list.
  If line element, L1, is in symbol, Si.
    Add symbol, Si to join list, PJoinL.
    Remove symbol Si from Symbol List.
  Process next symbol.
\normalfont
```

This search process is recognized as being inefficient, however is not currently presenting any process delays, refer section 4 for future implementations.

Once the search procedure is complete, the resulting symbol list is classified as being either:

1. **Shape List** - The Start Shape is required to join the end shape, thus constructing a completed shape object.

2. **Join List** - Joins are used to connect shape object together. A join object is determined is the following conditions are met:
    - Last two symbols have the same rotational angle.
    - Last symbol is a 90^o angle and the second last symbol is a 45^o angle.
3. **Unknown List** - This is any list of symbols which can not be classified as one of the previous types.

Once each symbol list has been generated and its type determined it is then ordered. The general ordering algorithm works by rotating the list until the symbol with the smallest *identifier* is the first object in the list. The list should then rotate so that the next smallest *identifier* in the list is as close as possible to the start of the list.

Some custom ordering algorithms have been implemented for special shape object such as join shapes and shapes containing the special Apostrophe symbol.

The ordering process is performed as it allows a simplified left-right HMM to be used, refer [5] for detailed explanation of HMM theory.

2.7. Join shape objects together using join objects

The symbols lists are then joined together to result in the HMM input. This input should have the following parts:

1. Start symbol to indicate the beginning of a new sequence.
2. Object symbol list.
3. Space symbol
4. Join symbol list
5. Space symbol.
6. Object symbol list.
7. End symbol to indicate the end of the sequence, this can only be followed by a new start symbol.

These hmm data structures are created by processing each Join symbol list using the following pseudo code:

```
While Join symbol list left to process
  For all Shape Objects.
    Find object touching start of join list
  If start of join list connects to object.
    For all shape objects.
      Find object touching end of join list
    If end of join list connects to object.
      OutputComplete shape object
      Remove join list from processing pile.
    Else
      For all unknown objects.
        Find object touching end of join list
      If end of join list connects to object.
        OutputComplete shape object
        Remove join list from processing pile.
      Else
        Mark as "'Processed-No Match"'
Else
  For all unknown Objects.
    Find object touching start of join list
  If start of join list connects to object.
    For all Shape Objects.
      Find object touching end of join list
    If end of join list connects to object.
      OutputComplete shape object
      Remove join list from processing pile.
    Else
      For all unknown objects.
        Find object touching end of join list
      If end of join list connects to object.
        OutputComplete shape object
        Remove join list from processing pile.
      Else
        Mark as "Processed-No Match"
```

The joining algorithm works by calculating the minimum distance between the start or end point of the join symbol list and one of the edges of the symbol list. A shape object is detected as connecting to the join object is this minimum distance if less than the system parameter, *minCornDist* (minimum corner join distance). If this does not produce a match then the unknown symbol list is also searched for a connecting shape object.

If the process is able to find a shape to connect with the start and end point then the symbol lists for these three symbol lists are concatenated with the appropriate system generated symbols; start symbol first; space symbol between shape and join objects and end symbol last.

2.8. Generating Transition Table and Most Likely State list

Once the symbol sequence has been generated it is passed to the HMM procedure. This produces a pstate transition table and most likely state transition sequence. The pstate transition table holds the probability of being any individual state are any position in the symbol sequence. The pstate transition table is then used to calculate the most likely state sequence.

2.9. Output Diagrams to specified format

Once the state sequence has been generated the resulting data is used to build a flow tree which represents the user input diagram. This representation is then used to extract exact parameters for each shape objects which allows the graphical representation to be updated. In addition this representation is used to output the diagram as a series of vector representation, which can then be imported into other programs.

3. RESULTS

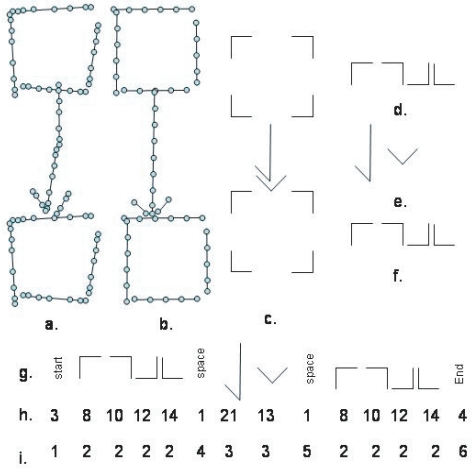

Fig. 5. Processing Stages; a) Representation of the users input stream; b) Normalized input stream with lien line type and rotation calculated; c) Representation of all generated systems; d) Shape symbol list for first shape object; e) Join symbol list; f) Shape symbol list for second shape object; g) Represenattion of joined symbol sequence; h) numerical representation of join symbol sequence to be passed to HMM decoder; i) The HMM determined most likely state transition (refer Section 3 for state defintions)

Training and testing of the HMM was performed using the Matlab [8] at QUT [9].

Initial results have been produced for a HMM system with the following characteristics:

- State Count (M) = 6
- State List: Start (1), Box (2), Down Arrow (3), First Space(4), Second (5) Space and Stop (6)
- Symbol Count (N) = 24 (8 system, 8 90^o Angle and 8 45^o Angle)
- Symbols in use: 1..4 (System Symbols) ,8..15 (90^o Angle), 16..23 (45^o Angle)
- Training Dataset Size: 500 Symbol sequence.

Using the trained model within the Matlab application, initial detection probabilities for 99.2% were achieved for simple data sets. These have proved to be encouraging results and implementation in an increased number of system states is currently beg implemented. Figure 5 illustrates teh processing stages which the system performs and a represntation of the data sets used. The output data shown domes from a simple HMM models without any input errors, as such the recognition rate is 100%.

4. CONCLUSION AND FUTURE WORK

HMM recognition models have been successfully implemented in an increasingly large number of applications. The work currently being performed to implement this recognition model within a less structured two dimensional environment such as diagram recognition has shown some initially promising results. It is expected that continued work on this application will result in a product which is able to efficiently perform diagram recognition and output in vector format in real-time.

The following items have been identified for future work:

1. Increase number of rotational angles as this will require larger more complex training sets, but will allow an increased range of differing systems to be produced.
2. Increase symbol set to include all possible angle combinations.
3. Increase system states to include all normal diagram input shapes and objects.
4. Add the ability to generate a symbols at a point where the end of one line joins to the middle of another straight line segments or two lines segments cross in their middles.
5. Include arc type lines as possible symbol generation elements, this will allow recognition of circles, ovals and other shapes incorporating rounded sections.
6. Implement the recognition algorithm to function in real-time so shapes and objects are recognized as they are completed where possible.
7. Improve the performance of search algorithms, especially in the symbol list generation phase where join knowledge can be generated during the symbol generation stage.

5. REFERENCES

[1] io^{TM} Logitech® *Personal Digital Pen*, http://www.logitechio.com/.

[2] Seiko Instruments USA Inc., *SmartPad for Pocket PC*, http://www.siibusinessproducts.com/products/sp580p.html.

[3] Wacom, *Gaphire2 Pen, Mouse and Tablet set - G-410 Series*, http://ap.wacom.co.jp /products/graphire/graphire2_index.html.

[4] Compaq, *Compac Tablet PC TC 1000*, http://h18000.www1.hp.com/products/tabletpc.

[5] R Schwartz T Starnert, J Makhoul and G Chou, "On-line cursive handwriting recognition using speech recognition methods," in *IEEE, BBN Systems and Technologies*, 1994.

[6] Manuel J. Fonseca Joaquim A. Jorge, "Experimental evaluation of an on-line scribble recognizer," *Pattern Recognition Letters*, vol. 22, no. 12, pp. 1311–1319, 10/2001.

[7] Citrin W.V. and M.D Gross, "Distributed architectures for pen-based input and diagram recognition," in *ACM Conference on Advanced Visual Interfaces '96*, 1996.

[8] Matlab Mathsworks, *Statistical Toolkit 4.1 for Matlab 6.5*, http://www.matlab.com.

[9] Computational (and/or data visualisation) resources, services used in this work were provided by the HPC, and Research Support Group, ," in *Queensland University of Technology, Brisbane, Australia*, 1996.