



UNIVERSITY OF  
ILLINOIS LIBRARY  
AT URBANA-CHAMPAIGN



The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

APR 03 1961  
LIBRARY USE ONLY



510.84  
IL6N  
no. 660

Math

9

Report No. UIUCDCS-R-74-660

ON-LINE CHARACTER RECOGNITION

by

Alfred C. Weaver

August 1974



DEPARTMENT OF COMPUTER SCIENCE  
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

SEP - 4 1974

THE LIBRARY OF THE

UNIVERSITY OF ILLINOIS

AT URBANA-CHAMPAIGN



Report No. UIUCDCS-R-74-660

ON-LINE CHARACTER RECOGNITION

by

Alfred C. Weaver

August, 1974

Department of Computer Science  
University of Illinois at Urbana-Champaign  
Urbana, Illinois 61801



Digitized by the Internet Archive  
in 2013

<http://archive.org/details/onlinecharacter660weav>



## TABLE OF CONTENTS

	<u>Page</u>
1. THE EVOLUTION OF ONLINE CHARACTER RECOGNITION . . . . .	1
1.1 Introduction . . . . .	1
1.2 History . . . . .	1
1.3 Essential Characteristics of Recognizers . . . . .	2
1.3.1 Hardware . . . . .	2
1.3.2. Software . . . . .	3
1.3.3 Feature Recognition . . . . .	4
1.4 Advantages . . . . .	6
1.5 Disadvantages . . . . .	6
1.6 Summary . . . . .	7
2. THE RECOGNITION ALGORITHM . . . . .	8
2.1 The ONLINE Simulation Program . . . . .	8
2.1.1 Theory . . . . .	8
2.1.2 Program Input . . . . .	9
2.1.3 Stroke Encoding . . . . .	9
2.1.4 The Dictionary . . . . .	11
2.1.5 Simulator Operation . . . . .	12
2.1.5.1 Training Mode Detail . . . . .	13
2.1.5.2 Recognition Mode Detail . . . . .	13
LIST OF REFERENCES . . . . .	14



APPENDIX

Page

1. A POSSIBLE STROKE SEQUENCE FOR THE ALPHANUMERIC CHARACTERS . . . . .	15
2. SOURCE CODE FOR 'ONLINE' SIMULATOR PROGRAM . . . . .	19
3. SAMPLE OUTPUT FROM 'ONLINE'. . . . .	24
4. SAMPLE OUTPUT FROM 'ONLINE'. . . . .	28



## LIST OF FIGURES

Figure	Page
1. A Voltage Gradient Tablet. . . . .	2
2. Stroke Sequences Encoded by Regions . . . . .	4
3. Listing of 4-bit Code Words . . . . .	10



## 1. THE EVOLUTION OF ON-LINE CHARACTER RECOGNITION

### 1.1 Introduction

The impetus for this project involving machine recognition of hand-printed characters was my dissatisfaction with the current state of on-line character recognition machinery and algorithms, as described in Chapter 12 of Principles of Interactive Computer Graphics [4], by Newman and Sproull. The set of criteria for the "perfect recognizer," as suggested by Newman and Sproull, is:

- (1) responds quickly;
- (2) high rate of success of recognition;
- (3) tolerates variation in size, style, and orientation;
- (4) uses computer resources sparingly.

The authors point out that no current (1973) recognizers can claim to meet all these criteria perfectly.

I believe that the hardware and software which I will propose will meet the above criteria as closely as any on-line character recognition machinery I have seen, while simultaneously maintaining a modest cost.

### 1.2 History

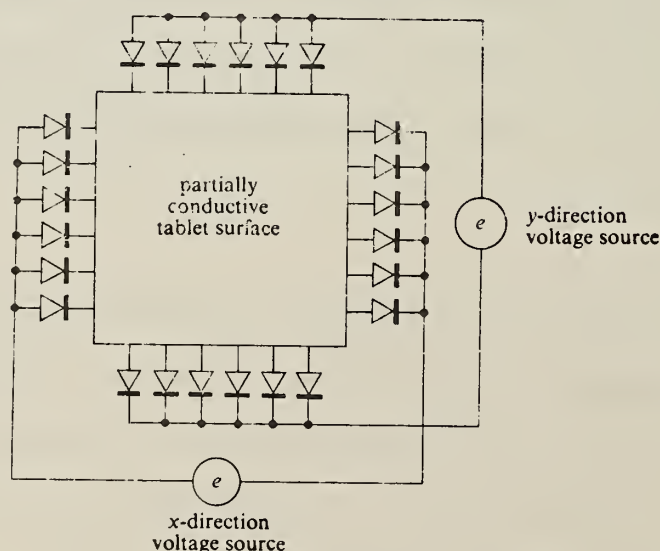
The first on-line character recognizers used graphic tablets and mini-computers as the hardware, and had very inflexible software (the user) could not train the machine to recognize "his" character, but rather the machine trained the user to adopt a predefined style). An example of just such a system is discussed by Groner [2].

The first trainable recognizer was developed by Teitelman [5]; others, including Bernstein [1], soon developed trainable programs utilizing the RAND tablet. While progress has not altogether stopped on such systems, the number of references to recent work (after 1970) is surprisingly and disappointingly small.

### 1.3 Essential Characteristics of Recognizers

#### 1.3.1 Hardware

To recognize the alphabetic, numeric, and special characters of, say, the ASCII code, as well as other special characters demanded by any particular system, a graphic tablet is essential. Previous work with RAND and Sylvania tablets indicates that a large portion of the hardware budget goes into the tablet itself. My proposal is to substitute a voltage gradient tablet of the form described by Newman and Sproull [4]:



A Voltage Gradient Tablet

Figure 1



The advantages of this choice are primarily its lower cost, secondarily its inherent ability to be fine-tuned in the field, and thirdly the small amount of additional (and expensive) hardware which must be added to the basic tablet to realize a useful input device.

In addition to the voltage sources shown in Figure 1, the system needs simple circuitry to recognize an interrupt from the stylus (making or breaking contact with the tablet surface), an A/D converter, an interrupt flip-flop, and two 10-bit X and Y position registers.

The necessary computing can be handled entirely by a microprocessor of the Intel 8008 variety (1974 prices are approximately \$100 for the chip alone, or \$900 for the MCS-8, which includes the 8008 chip, all other necessary circuitry and 3K memory). I chose this particular microcomputer because I was familiar with its capabilities and know it could easily be adapted for this system.

### 1.3.2 Software

The basic requirements of the software system are:

- (1) read the tablet;
- (2) extract important features from character;
- (3) dictionary lookup routine;
- (4) training routine to build dictionary.

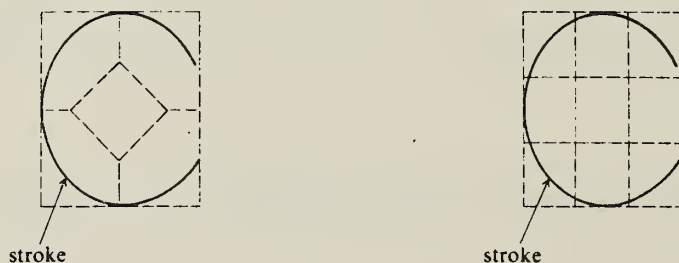
All of the above are presently in my simulator program ONLINE. I am convinced that the entire program will fit with room to spare on an MCS-8.

### 1.3.3 Feature Recognition

The crucial part in the design of the recognizer is the choice of features to be extracted from the input strokes. A good recognizer must discriminate between different characters while recognizing slightly different versions of the same character.

The most popular features extracted by current mechanisms are the regions visited by each stroke. By predefining a standard rectangle, or by normalizing the stroke to some standard rectangle, and then dividing that rectangle into a number of regions, a program may then count and record the number of times a stroke crosses a region boundary.

This is exactly the scheme used by both Bernstein [1] and Teitelman [5], as shown in Figure 2.



Stroke Sequences Encoded by Regions

Figure 2

After collecting a stroke sequence, these schemes searched a tree-structured dictionary, yielding the character to be recognized. Ledeen [3] developed a simple recognizer which used a similar feature extraction mechanism, but utilized a more compact dictionary (about 1K 16-bit words).

My objections to these known schemes are three-fold:

- (1) The use of RAND-type tablets is too costly;
- (2) I disagree in theory with having to write into a predefined rectangle, using predefined region boundaries. The alternative, "clipping" to the boundaries of the character itself and then subdividing into regions, only increases the amount and cost of processing power and memory needed, and hence is equally unsatisfactory.
- (3) The tree-structured dictionary uses more memory than necessary. If the dictionary is actually a tree, with nodes and pointers, approximately  $2/3$  of the memory space is devoted to pointers and only  $1/3$  to data. If the dictionary is a binary tree, but structured as an array [left son of node  $n$  at  $T(2n)$  and right son at  $T(2n+1)$ ], then the total array area must grow in proportion to the depth of the tree, i.e., the longest stroke sequence. Since adding one level to the tree will double the array area required, effective memory utilization remains a problem.

My solutions to the three objections above are:

- (1) Use a resistive tablet. This relatively low cost input device contains all the sophistication necessary for character recognition;
- (2) Make the recognition process independent of the size and location of the character drawn. This is accomplished by a new definition of the "essential features" of characters: the order of strokes and their direction, independent of their physical location;

- (3) Encode the stroke sequence and orientation into one code number (explained in Algorithm) and look up the code number in a sorted dictionary. The code number has the same information content as the tree traversal order, without the memory overhead.

The whole scheme is made viable by using the classical approach of having an initial training period in which the user trains the machine to recognize his own writing style (in fact, the user may teach several stroke sequences for the same character). Following the training period the machine runs in its normal recognition mode. Provision is made for retraining the machine when another user takes over.

#### 1.4 Advantages

The worth of the solution described here is enhanced by its direct applicability to a microprocessor environment, its sparing use of costly memory, and the small amount of interface hardware necessary between the tablet and processor. It allows the user to train the machine to his own writing style. Different users may operate the machine by utilizing a retraining mode. Furthermore, the entire algorithm is programmable using only addition, subtraction, and testing. No multiplications or divisions are required.

#### 1.5 Disadvantages

It was first thought that the requirement of distinct strokes would allow recognition of only block-style printed characters. However, the scheme generalized quite nicely to the normal spectrum of printed

characters, with only a very few unnatural cases caused by describing two different characters by the same stroke pattern.

## 1.6 Summary

I believe that the "Weaver method" represents an elegant solution to an otherwise sticky problem. Like its predecessors, this method fails to fully satisfy all of Newman and Sproull's basic criteria for the perfect recognizer. Yet, it satisfies sufficiently many of them that it would perform well in many common situations. Its most significant advantages are its low cost and ease of implementation.

Appendix 1 contains a list of the alphanumeric characters and their stroke sequence definitions for the author's style of writing. Appendix 2 is a listing of the computer program which simulates the hardware and software systems. Appendix 3 is a sample run which shows the training and recognition of all 36 alphanumeric characters. Appendix 4 illustrates the change of mode, error detection, and symbol non-recognition.

## 2. THE RECOGNITION ALGORITHM

### 2.1 The ONLINE Simulation Program

The program ONLINE simulates the actions of:

- (1) A resistive tablet and stylus;
- (2) The hardware interface between tablet and processor, including X- and Y-position 10-bit registers and 1-bit interrupt flag;
- (3) The processor (a microprocessor, such as the Intel 8008).

This simple equipment is shown to be sufficient for an on-line character recognition system.

#### 2.1.1 Theory

The touching of the stylus to the tablet generates an interrupt and sets an interrupt flag. Upon seeing this flag the processor copies the contents of the position registers into its variables  $X_1$  and  $Y_1$ . The lifting of the stylus from the tablet also generates an interrupt and sets an interrupt flag. This time the processor copies the position registers into its variables  $X_2$  and  $Y_2$ . Now by simple subtraction and testing, the processor determines a four-bit code word for each line  $(X_1, Y_1) \rightarrow (X_2, Y_2)$  drawn.

The code is of the form  $C_1 C_2 C_3 C_4$ ,

where  $C_1$  is 1 only if the line was drawn from top to bottom,  
 $C_2$  is 1 only if the line was drawn from bottom to top,  
 $C_3$  is 1 only if the line was drawn from left to right,  
 and  $C_4$  is 1 only if the line was drawn from right to left.



The code word is set by a program segment similar to this:

```

C1 = C2 = C3 = C4 = 0;

if (Y1-Y2) > EPSILON then C1 = 1;
    else if (Y2-Y1) > EPSILON then C2 = 1;

if (X2-X1) > EPSILON then C3 = 1;
    else if (X1-X2) > EPSILON then C4 = 1;

```

### 2.1.2 Program Input

The input to the simulator program for any one character is a series of "strokes" which represent the hardware result of the previous algorithm. For instance, the letter "A" is defined by three strokes:

<u>stroke number</u>	<u>stroke orientation</u>	<u>stroke encoding</u>
1	top to bottom & right to left	TB,RL/
2	top to bottom & left to right	TB,LR/
3	left to right	LR/

Each stroke is encoded by using the obvious 2-character mnemonic for each of the four possible directions, with a "/" used to indicate the end of one stroke. Thus the full description of "A" is:

A: TB,RL/TB,LR/LR/

### 2.1.3 Stroke Encoding

From this description, provided as a character string on an input card to the simulator, the processor decodes the input into sequence of 4-bit code words, one code word per stroke. Now, by using a binary digit weighting scheme on the 4-bit words, a unique hexadecimal (0-15) digit results.

Note that, of the 16 possible codes, only 8 are used (combinations like top-to-bottom and bottom-to-top are clearly impossible). The usable combinations are shown in Figure 3.

$C_1$ (TB)	$C_2$ (BT)	$C_3$ (LR)	$C_4$ (RL)	Hexadecimal Equivalent
0	0	0	1	1
0	0	1	0	2
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10

Listing of 4-bit Code Words

Figure 3.

Thus any sequence of strokes is reducible to a series of hex digits. "A" is encoded as 9/10/2/.

Since 10 is the largest hex digit used, subtracting one from each of the hex equivalents in Figure 3 provides a corresponding set of decimal digits which are more easily handled. The decimal digits corresponding to the stroke sequence are then weighted with a power of ten, proportional to their positions in the stroke sequence, to yield a single decimal number as the representative code number. Repeating the example of the example of the character "A",



A	TB,RL/TB,LR/LR/	9/10/2	8/9/1	891
character	simulator input	hex	decimal	code number

Clearly, this process may be run backwards to generate the defining stroke sequence which generated any given code number.

The worst case for the length of a stroke sequence for common alphanumeric characters appears to be 4 strokes. Even allowing for 5 strokes, the corresponding code number is less than  $10^5$ , and so remains well within the range of microcomputer arithmetic.

While the above scheme works quite well, generalizing from block characters to printed characters required one minor modification - recognition of the "null" stroke generated by curves which begin and end within EPSILON of the same point, as in the letter "O", the number "ø", the letter "Q", and the numbers "6", "8", and "9". This was easily accomplished by allowing a previously unused decimal digit (6) to mark a "null" stroke.

#### 2.1.4 The Dictionary

Now that the input stroke sequence has been encoded into a few bits, it appears to be best managed by storing the decimal code numbers and the characters which they represent in a dictionary, formed from two linear arrays, and arranged in sorted order by ascending code numbers. The beginning of a typical dictionary might be:

071	C
711	F
891	A
⋮	

A binary search on the code numbers suffices to quickly locate any character in the list. In recognition mode this is exactly what happens; in training mode the addition of a new code-number-and-character combination uses the binary search to find their proper position in the list. The list is then shifted to make a hole, and the new information is inserted. The efficiency of the binary search is well known; for a list of, say, 64 characters, the maximum number of probes required to locate any character is given by  $\log_2 64 = 6$ .

#### 2.1.5 Simulator Operation

The simulator works in one of five modes:

Training: A character is presented and its stroke sequence defined.

Example: A: TB,RL/TB,LR/LR/

Z: LR/TB,RL/LR/

Recognition: Only the stroke sequence is presented; it is decoded and its associated character printed.

Example: TB/TB/LR/LR/

Restart: Clears all currently stored information in preparation for a new training sequence.

Ignore:            In the event of invalid commands, they are ignored.

Stop:             Program terminates.

#### 2.1.5.1 Training Mode Detail

The first character of the input card contains the new character to be learned. The remaining characters are the stroke sequence mnemonics which the user wishes to associate with this character. The stroke sequence is decoded into its code number n, the current (sorted) list of code numbers is binary searched to find this number's proper location, all entries whose code number is larger than n are moved down, and the current code number and character are inserted.

#### 2.1.5.2 Recognition Mode Detail

Each input card contains only the simulated stroke sequence. The sequence is decoded into its corresponding code number and the current dictionary is searched. If the code number is found, its associated character is echoed along with the message STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "\*", where \* is the recognized character. If the code number is not found, the message "STROKE SEQUENCE NOT RECOGNIZED. TRY AGAIN" appears. If, when the stroke sequence is repeated, it is still not recognized, the message "STILL NOT RECOGNIZED. RETRAIN FOR THIS SYMBOL" is produced. It is probable that the user has now either changed his defining stroke sequence for a particular character, or used a new symbol unfamiliar to the recognizer. In either case, the recognizer should return to training mode to learn a new stroke sequence; this is speedily accomplished with the \$TRAIN command.



































## LIST OF REFERENCES

1. Bernstein, M.I., and T. G. Williams, "A Two-Dimensional Programming System," Proceedings 1968 IFIP Congress, North Holland Pub. Co., page 586, 1969.
2. Groner, G. F., "Real-Time Recognition of Hand Printed Text", FJCC 1966, Spartan Books, page 591.
3. Ledeen, K. S., "The Ledeen Character Recognizer," Principles of Interactive Computer Graphics, appendix VIII, 1973.
4. Newman, W. M., and R. F. Sproull, Principles of Interactive Computer Graphics, McGraw-Hill Book Co., 1973.
5. Teitelman, W., "Real Time Recognition of Hand-Drawn Characters," FJCC 1964, Spartan Books, page 559.

APPENDIX 1

The alphanumeric characters and their defining stroke sequences, as determined by the author's style of writing.

A POSSIBLE STROKE SEQUENCE FOR THE  
ALPHANUMERIC CHARACTERS

	first stroke	second stroke	third stroke	fourth stroke	decimal code number
A					891
B					777
C					7
D					77
E					7111
F					711
G					91
H					717
I					171
J					81
K					789
L					9
M					8989

	first stroke	second stroke	third stroke	fourth stroke	decimal code number
N	↓	↓	↓		797
O	○				6
P	Р				3
Q	○	↓			69
R	Р	↓			39
S	S				8
T	↓	→			71
U	↻				1
V	↓	↙			98
W	↓	↙	↓	↙	9898
X	↙	↓			89
Y	↓	↙	↓		987
Z	→	↙	→		181
ø	○	↙			68
1	↓	→			78
2	2	↓			97
3	→	↙	↻		188

	first stroke	second stroke	third stroke	fourth stroke	decimal code number
4	↘	↓			87
5	↓	↙	→		781
6	↓	○			76
7	→	↙			18
8	○	○			66
9	○	↓			67



APPENDIX 2

Source code listing for the simulator program ONLINE.

```

/* A SIMULATOR OF ON-LINE CHARACTER RECOGNITION */
ONLINE: PROCEDURE OPTIONS(MAIN);

```

```

/* A PROJECT FOR CS 397, INTERACTIVE COMPUTER GRAPHICS (C. W. GEAR)
PROGRAMMED BY ALFRED C. WEAVER
DATE: MAY 4, 1974

```

THIS PROGRAM SIMULATES A MICROPROCESSOR SUCH AS THE INTEL 8008 WITH A VOLTAGE GRADIENT (RESISTIVE) TABLET AS THE INPUT DEVICE. THE ALGORITHM RECOGNIZES HAND-WRITTEN CHARACTERS BY EXTRACTING ESSENTIAL FEATURES (STROKE SEQUENCE AND ORIENTATION) FROM THE INPUT. STROKE SEQUENCES ARE ENCODED AS A DECIMAL CODE NUMBER WHICH SERVES AS REFERENCE IN A SORTED, LIST-STRUCTURED DICTIONARY.

THE RECOGNIZER OPERATES IN BOTH "TRAINING" AND "RECOGNITION" MODES. WHILE IN TRAINING THE RECOGNIZER ENCODES STROKE SEQUENCES AND BUILDS A DICTIONARY. WHEN RECOGNIZING THE INPUT STROKES ARE ENCODED AND THE DICTIONARY SEARCHED FOR THE CORRESPONDING CHARACTER. PROVISION IS MADE WITH THE "RESTART" MODE TO CLEAR MEMORY AND BEGIN TRAINING BY A DIFFERENT USER.

THE ALGORITHM IS DESIGNED FOR A MICROPROCESSOR ENVIRONMENT WITH LITTLE ADDITIONAL INTERFACE HARDWARE. THE PRIMARY ADVANTAGES ARE LOW COST, SIMPLICITY OF PROGRAMMING, AND EASE OF USE. \*/

```

/* PREPARE THE MICROPROCESSOR DATA AREA */
DCL RCHAR(1:64) CHAR(1),      /* LIST OF RECOGNIZED CHARACTERS */
    NEWCHAR CHAR(1),          /* NEW CHARACTER IN TRAINING MODE */
    CARD CHAR(80) VAR,        /* INPUT CARD IMAGE */
    FOUND BIT(1),             /* LOCATE FLAG */
    (POINTER,                 /* LOCATION POINTER SET BY 'LOOKUP' */
     #ENT,                     /* CURRENT NUMBER OF TABLE ENTRIES */
     CODE#,                    /* ENCODED STROKE SEQUENCE */
     NOTFOUND,                 /* COUNT OF INPUT ERRORS */
     LIST(1:64),               /* LIST OF RECOGNIZED CODE WORDS */
     MODE) FIXED BINARY(31);   /* MODE OF OPERATION:
                                =0  IGNORE
                                =1  TRAIN
                                =2  RECOGNIZE
                                =3  STOP
                                */

```

```

/* INITIALLY CLEAR THE DATA AREA */
CALL RESTART;

/* REPEAT UNTIL $STOP COMMAND IS PROCESSED */
DO WHILE (MODE  $\neq$  3 /* STOP */);

/* READ AND PRINT INPUT CARD */
GET EDIT (CARD) (COL(1), A(30));
PUT SKIP(2) EDIT (CARD) (A);

/* IF COLUMN 1 IS $, THEN CARD IS A COMMAND */
IF SUBSTR(CARD,1,1) = '$' THEN

    /* SET PROPER MODE */
    IF CARD = '$TRAIN' THEN MODE = 1;
    ELSE IF CARD = '$RECOGNIZE' THEN MODE = 2;
    ELSE IF CARD = '$STOP' THEN MODE = 3;
    ELSE IF CARD = '$RESTART' THEN CALL RESTART;
    ELSE MODE = 0;

/* OTHERWISE, CARD IS AN INPUT STROKE SEQUENCE */
ELSE DO;
    /* DECODE THE STROKE SEQUENCE */
    CALL DECODE;
    /* FIND CODE WORD IN TABLE */
    CALL LOOKUP;
    /* WHICH MODE ARE WE IN? */
    IF MODE = 1 THEN /* IN TRAINING MODE */ CALL INSERT;
    ELSE /* IN RECOGNITION MODE */ DO;
        IF FOUND /* ALREADY IN DICTIONARY */ THEN PUT EDIT
        ('STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "',
        RCHAR(PJINTER), '"') ( 3 A);
        ELSE /* NOT IN DICTIONARY */ DO;
            IF NOTFOUND = 1 /* FIRST TIME */ THEN PUT EDIT
            ('CHARACTER NOT RECOGNIZED. TRY AGAIN') (SKIP, A);
            ELSE /* SECOND TIME NOT FOUND */ PUT EDIT
            ('STILL NOT RECOGNIZED. RETRAIN FOR THIS SYMBOL')
            (SKIP, A);
        END;
    END;
END;
END;
END;

PUT SKIP(3) EDIT ('END OF PROGRAM') (A);
RETURN;

RESTART: PROCEDURE;
/* A PROCEDURE TO CLEAR THE DATA AREA */
DECLARE I FIXED BINARY (31);
POINTER, #ENT, CODE#, MODE, NOTFOUND = 0;
DO I = 1 TO 64;
    LIST(I) = 0;
    RCHAR(I) = ' ';
END;

```

END RESTART;

```
LOOKUP: PROCEDURE;
/* A PROCEDURE TO FIND THE LOCATION OF 'CODE#' IN 'LIST' */
DECLARE (HEAD INIT(0), TAIL, MID, #2 INIT(2)) FIXED BINARY (31);
TAIL = #ENT + 1;
FOUND = '0'B;
```

```
/* STANDARD BINARY SEARCH */
DO WHILE (TAIL-HEAD > 1);
  MID = (HEAD+TAIL)/#2;
  IF CODE# < LIST(MID) THEN TAIL = MID;
  ELSE IF CODE# > LIST(MID) THEN HEAD = MID;
  ELSE DO;
    FOUND = '1'B;
    NOTFOUND = 0;
    POINTER = MID;
    RETURN;
  END;
```

```
  END;
POINTER = HEAD;
NOTFOUND = NOTFOUND + 1;
END LOOKUP;
```

```
INSERT: PROCEDURE;
/* A PROCEDURE TO INSERT 'CODE#' AND 'NEWCHAR' INTO DICTIONARY */
DECLARE I FIXED BINARY (31);
IF FOUND THEN RETURN;
POINTER = POINTER + 1;
DO I = #ENT TO POINTER BY -1;
  LIST(I+1) = LIST(I);
  RCHAR(I+1) = RCHAR(I);
END;
LIST(POINTER) = CODE#;
RCHAR(POINTER) = NEWCHAR;
#ENT = #ENT + 1;
END INSERT;
```

```
DECODE: PROCEDURE;
/* A PROCEDURE TO READ STROKE SEQUENCE AND PRODUCE 'CODE#' */
DECLARE FIELD CHAR(10) VAR, (I,C) FIXED BINARY(31);
/* IF IN TRAINING MODE, EXTRACT NEW CHARACTER */
IF MODE = 1 THEN DO;
  NEWCHAR = SUBSTR(CARD,1,1);
  CARD = SUBSTR(CARD,3);
END;
CODE# = 0;
/* REPEAT FOR EACH STROKE */
DO WHILE (CARD ^= ' ');
  I = INDEX(CARD, '/');
  /* CATCH INPUT FORMAT ERRORS */
  IF I = 0 THEN DO;
```

```

/* STOP ON INPUT FORMAT ERROR */
PUT SKIP EDIT ('INPUT FORMAT ERROR') (A);
STOP;
END;
ELSE DO;
  FIELD = SUBSTR(CARD, 1, I);
  CARD = SUBSTR(CARD, I+1);
  C = 0;
  IF INDEX(FIELD, 'TB') > 0 THEN C=C+8;
  IF INDEX(FIELD, 'BT') > 0 THEN C=C+4;
  IF INDEX(FIELD, 'LR') > 0 THEN C=C+2;
  IF INDEX(FIELD, 'RL') > 0 THEN C=C+1;
  IF C = 0 THEN C = 7;
  /* IN THE MICROPROCESSOR CODE THIS MULTIPLICATION MAY BE
     REPLACED BY ADDITIONS */
  CODE# = CODE# * 10 + C - 1;
  END;
END;
END DECODE;

END ONLINE;

```

APPENDIX 3

Sample output from ONLINE showing training and recognition  
for the 36 aphanumeric characters.

## \$TRAIN

A: TB,RL/TB,LR/LR/

B: TB/TB/TB/

C: TB/

D: TB/TB/

E: TB/LR/LR/LR/

F: TB/LR/LR/

G: TB,LR/LR/

H: TB/LR/TB/

I: LR/TB/LR/

J: TB,RL/LR/

K: TB/TB,RL/TB,LR/

L: TB,LR/

M: TB,RL/TB,LR/TB,RL/TB,LR/

N: TB/TB,LR/TB/

O: /

P: BT/

Q: /TB,LR/

R: BT/TB,LR/

S: TB,RL/

T: TB/LR/

U: LR/

V: TB,LR/TB,RL/

W: TB,LR/TB,RL/TB,LR/TB,RL/

X: TB,RL/TB,LR/

Y: TB,LR/TB,RL/TB/

Z: LR/TB,RL/LR/

0: /TB,RL/

1: TB/TB,RL/



2: TB,LR/TB/

3: LR/TB,RL/TB,RL/

4: TB,RL.TB/

5: TB/TB,RL/LR/

6: TB//

7: LR/TB,RL/

8: //

9: /TB/

\$RECOGNIZE

TB,RL/TB,LR/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "A"

TB/TB/TB/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "B"

TB/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "C"

TB/TB/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "D"

TB/LR/LR/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "E"

TB/LR/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "F"

TB,LR/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "G"

TB/LR/TB/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "H"

LR/TB/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "I"

TB,RL/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "J"

TB/TB,RL/TB,LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "K"

TB,LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "L"

TB,RL/TB,LR/TB,RL/TB,LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "M"

TB/TB,LR/TB/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "N"

/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "O"

BT/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "P"

/TB,LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "Q"

BT/TB,LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "R"

TB,RL/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "S"

TB/LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "T"

LR/

STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "U"



TB,LR/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "V"
TB,LR/TB,RL/TB,LR/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "W"
TB,RL/TB,LR/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "X"
TB,LR/TB,RL/TB/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "Y"
LR/TB,RL/LR/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "Z"
/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "0"
TB/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "1"
TB,LR/TB/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "2"
LR/TB,RL/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "3"
TB,RL.TB/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "S"
TB/TB,RL/LR/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "5"
TB//	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "6"
LR/TB,RL/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "7"
//	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "8"
/TB/	STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "9"
\$STOP	
END OF PROGRAM	

APPENDIX 4

Sample output from ONLINE showing change of mode,  
error detection, and symbol non-recognition.

\$RECOGNIZE

TB/BT/TB/  
CHARACTER NOT RECOGNIZED. TRY AGAIN

\$TRAIN

A: TB,RL/TB,LR/LR/

B: TB/TB/TB/

C: TB/

D: TB/TB/

E: TB/LR/LR/LR/

F: TB/LR/LR/

\$PECOGNIZE

TB/LR/LR/LR/ STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "E"

TB,PL/TB,LR/LR/ STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "A"

TB/TB/LR/LR/  
CHARACTER NOT RECOGNIZED. TRY AGAIN

TB/TB/LR/LR/  
STILL NOT RECOGNIZED. RETRAIN FOR THIS SYMBOL

\$TRAIN

#: TB/TB/LR/LR/

\$RECOGNIZE

TB/TB/LR/LR/ STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "##"

\$PESTART

\$TRAIN

0: /TB,PL/

1: TB/TB,RL/

2: TB,LR/TB/

3: LR/TB,RL/TB,RL/

4: TB,RL/TB/

5: TB/TB,RL/LR/

6: TB//

7: LR/TB,RL/

```
8:  //
9:  /TB/

$RECOGNIZE
    TB,LF/TB/          STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "2"
    TB//              STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "6"
    /TB/              STROKE SEQUENCE RECOGNIZED AS THE CHARACTER "9"

$BADCCOMMAND
$BADCCOMMAND
$RECOGNIZE
    TB,LF,TB,RL
INPUT FORMAT ERROR
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-74-660	2.	3. Recipient's Accession No.
	4. Title and Subtitle  On-line Character Recognition		5. Report Date August 1974
7. Author(s) Weaver, A.C.		8. Performing Organization Repr. No. UIUCDCS-R-74-660	
9. Performing Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. Project/Task/Work Unit No.	
		11. Contract/Grant No.	
12. Sponsoring Organization Name and Address Department of Computer Science University of Illinois at Urbana-Champaign Urbana, Illinois 61801		13. Type of Report & Period Covered	
		14.	
15. Supplementary Notes			
16. Abstracts  This report discusses the current state-of-the-art in computer recognition of handwritten characters. Several current schemes are examined and criticized. A new recognition method is developed utilizing a voltage gradient tablet for input and clever software for essential feature extraction. A simulation program is included as an appendix.			
17. Key Words and Document Analysis. 17a. Descriptors  on-line character recognition voltage gradient tablet essential feature extraction simulation			
17b. Comments. Open-Ended Terms			
17c. COSATI Field/Group			
18. Distribution Statement  release unlimited		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 30
		20. Security Class (This Page) UNCLASSIFIED	22. Price no charge











FEB 17 1981



UNIVERSITY OF ILLINOIS-URBANA



3 0112 050250510