

**ONLINE HANDWRITTEN MATHEMATICAL
EXPRESSION RECOGNITION**

by

HAKAN BUYUKBAYRAK

Submitted to the Graduate School of Engineering and Natural Sciences

in partial fulfillment of

the requirements for the degree of

Master of Science

Sabanci University

Spring 2005

ONLINE HANDWRITTEN MATHEMATICAL
EXPRESSION RECOGNITION

APPROVED BY

Prof. Dr. Aytul ERCIL
(Thesis Co-Supervisor)

Prof. Dr. Berrin Yanikoglu
(Thesis Co-Supervisor)

Prof. Dr. Alev TOPUZOGLU

Assist. Prof. Hakan ERDOGAN

Assist. Prof. Dr. Yucel SAYGIN

DATE OF APPROVAL:

©Hakan Buyukbayrak 2005

All Rights Reserved

Acknowledgments

I would like to thank my advisors Prof. Dr. Aytul Ercil and Prof. Dr. Berrin Yanikoglu for their guidance, encouragement, understanding throughout this study and also for providing the motivation and the resources for this research to be done.

I am also grateful to Prof. Dr. Alev Topuzoglu, Assist. Prof. Hakan Erdogan, Assist. Prof. Yucel Saygin for their participation in my thesis committee.

Special thanks to Ece Bagatur for all the encouragement and support she has provided throughout this thesis, in particular during the final stages of writing.

ONLINE HANDWRITTEN MATHEMATICAL EXPRESSION RECOGNITION

Abstract

This thesis presents a system for online handwritten mathematical expression recognition that involves integrals, summation notation, superscripts and subscripts, square-roots, fractions, trigonometric and logarithmic functions; together with a user-interface for writing scientific articles.

The aim of this study is to utilize the most convenient man-machine-interface, a pen, for input of mathematical expressions. In pen-enabled devices, handwriting sequences are collected by the digitization of pen movements which outputs an array of coordinates called strokes.

A neural network is trained for recognizing each stroke and a recursive algorithm parses the expression by combining neural network output and structure of the expression.

The interface associated with the proposed system integrates the built-in recognition capabilities of the Microsoft's Tablet PC-API for recognizing textual input and also supports conversion of hand-drawn figures into PNG format, which enable the user to enter text, mathematics and draw figures in a single interface. After the recognition, all output is combined into one \LaTeX code and compiled into a PDF file.

The system presented in this thesis provides a natural interface, hence enables easy-input of mathematical expressions in all pen-enabled devices such as tablet PCs, PDAs, external tablet pads, electronic pen-boards etc.

ÇEVİRİMİÇİ EL YAZISI MATEMATİK İFADE TANIMA

Özet

Bu çalışma kalem ile giriş yapılabilen tablet PC, PDA, dışarıdan bağlanan kalemli padler, elektronik yazı tahtaları gibi aygıtlarda yazılan el yazısı matematik denklemlerin algılanmasını sağlayacak sistemin isterlerini ve parçalarını ele almaktadır.

Matematik ifadeleri tanıyabilecek bir sistem integral, bölüm, üstler, indisler, karekökler, toplam sembolü vs. gibi matematiksel yapıları tanıyabilmelidir. Kağıt üzerine kalem ile yazarak bu yapıların hepsini kolayca belirtmemiz mümkün olduğu halde, şu ana kadar bilgisayara bu yapıları tanımlamak için yeterince kolay bir metod geliştirilememiştir. Yukarıda saydığımız aygıtlar ve bu çalışmada önerdiğimiz metod ile bilgisayar ortamında da yeterince kolay bir şekilde matematik yapıların tanımlanabilmesi sağlanmıştır.

Bu aygıtlarda el yazısı dizisini elde etmek için bir kalem kullanılmaktadır. Bu kalemin çıktısının sayısallaştırılması ile, kalemin yazmaya başlaması ve yazmayı bitirmesi arasındaki noktaların koordinatları ve bu koordinatlara ait zaman bilgileri elde edilmektedir. Her bir kalem darbesi programımızın içerisinde bir koleksiyonda tutulmaktadır.

Her bir kalem darbesi bir yapay sinir ağından geçirilmekte ve bu ağdan gelen sembol bilgisi, denklemin yapısal bilgisi ile birleştirilerek recursive bir okuyucu modül tarafından okunmaktadır.

Bu çalışmada önerilen sistemin arayüzü, aynı zamanda Microsoft'un Tablet PC-API'si içerisinde bulunan el yazısı tanıma modülünü de kullanmakta ve bu sayede hem matematik, hem de yazı girişini mümkün kılmaktadır. Bu sayede, tek bir arayüzde, hem matematik hem yazı içeren sayfaların oluşturulabilmektedir. Tanıma ve okuma işlemleri tamamlandığında, tüm çıktılar birleştirilerek tek bir \LaTeX kodu oluşturulmakta ve bir PDF dosyası üretilmektedir.

Table of Contents

Acknowledgments	iv
Abstract	v
Özet	vii
1 Introduction	1
1.1 Motivation	1
1.2 Previous Work	2
1.3 Overview	3
2 Math Symbol Recognition	4
2.1 Math Symbol Set	4
2.1.1 \LaTeX Math Symbols	4
2.1.2 Selected Symbols for Online Recognition	5
2.1.3 Symbol Classification Methods	5
2.2 Neural Networks	6
2.3 Data Collection	8
2.3.1 Pen-Input for Symbols	9
2.3.2 Normalization of Pen-Input	9
2.4 Classification of Individual Symbols	10
3 Expression Parsing	12
3.1 Mathematical Structure	12
3.2 Mathematical Expressions in \TeX	13
3.3 Parsing Simple Structures	14
3.3.1 Fractions	14
3.3.2 Superscripts and Subscripts	15
3.3.3 Integral Parsing	16
3.3.4 Square Root Parsing	16

3.3.5	Summation Parsing	16
3.3.6	Other Symbols and Notations	17
3.4	Multiple Structures	17
3.4.1	Recursive Parser	18
3.4.2	Parsing Example	18
4	Article Structure Recognition	21
4.1	Article Structure	21
4.2	Combining Symbol & Characters	22
4.2.1	Word Grouping	22
4.2.2	Line Grouping	23
4.2.3	Paragraph Grouping	24
4.3	Handling Figures and Expressions	24
5	System Specifications & User Interface	25
5.1	General Platform Information	25
5.1.1	Microsoft Tablet PC Application Programming Interface	25
5.1.2	Halcon Library	26
5.1.3	Visual Studio .NET & C#	26
5.2	Interfaces	27
5.2.1	Sample Collection Interface	27
5.2.2	Math Only Interface	27
5.2.3	Text & Figure & Math Interface	29
6	Conclusions & Future Work	32
	Appendix	34
A	Examples of Parsed & Recognized Expressions	34
B	Examples of Parsed & Recognized Articles	37
C	ℒ_TE_X Symbols	40
	Bibliography	45

List of Figures

2.1	Single Stroke Equivalents of 66 Symbols	6
2.2	Ink Collection - Points	9
2.3	Sub-Sampling (a) X Samples, (b) Y Samples, (c) Both Samples . . .	10
2.4	Neural Network Classifier Correct Classification Rate Graph	11
3.1	Fraction Parsing	15
3.2	Superscript and Subscript Parsing	16
3.3	Integral Parsing	16
3.4	Square Root Parsing	17
3.5	Summation Parsing	17
3.6	Multiple Structure Example	18
3.7	Parsing Methodology	19
4.1	Basic Article Structure	22
4.2	Word Grouping	23
4.3	Line Grouping	23
4.4	Paragraph Grouping	24
5.1	Sample Collection Interface	28
5.2	Math Only Interface	29
5.3	Math Only Interface - Matrix Mode	30
5.4	Text & Figure & Math Interface	31
5.5	PDF Output	31
A.1	Expression Example 1	34

A.2	Expression Example 2	34
A.3	Expression Example 3	34
A.4	Expression Example 4	35
A.5	Expression Example 5	35
A.6	Expression Example 6	35
A.7	Expression Example 7	35
A.8	Expression Example 8	36
A.9	Expression Example 9	36
A.10	Expression Example 10	36
A.11	Expression Example 11	36
A.12	Expression Example 12	36
B.1	Article Example 1	37
B.2	Article Example 2	38
B.3	Article Example 3	38
B.4	Article Example 4	39

List of Tables

2.1	66 Available Symbols	5
C.1	Greek Letters	40
C.2	Binary Operation Symbols	41
C.3	Relation Symbols	41
C.4	Punctuation Symbols	41
C.5	Arrow Symbols	42
C.6	Miscellaneous Symbols	42
C.7	Variable-sized Symbols	43
C.8	Log-like Symbols	43
C.9	Delimiters	43
C.10	Large Delimiters	43
C.11	Math mode accents	43
C.12	Some other constructions	44

Chapter 1

Introduction

1.1 Motivation

The problem of recognition of handwritten has long been a focus of study [1,15]. With the development of faster computers and increasing number of pen-enabled devices, the research in this area is again gaining focus. The handwritten input is a natural way of interacting with a computer and possibilities of different inputs is much higher than a keyboard. A pen can be used for writing text, drawing figures, clicking on a button, writing a complex equation even for playing a game. The desire for utilizing the pen input drives the research in this area.

The spread of pen-enabled devices started with PDAs. With a pen and a special alphabet it was possible to replace a keyboard. These devices did not have enough computing power for higher level machine recognition, however, with the recent PDAs, there are enough resources for handling recognition tasks.

In 2002, Microsoft released a version of Windows XP for Tablet PCs which triggered increasing amount of sales for tablet PCs. These PCs have a regular CPU like any laptop, and a pen-interface. Microsoft also released a Tablet PC programming platform which provided easy access to pen and pen programming.

As a result, there are increasing number of applications for natural handwritten input and commercially successful applications which let people to manage their appointments, memos, take notes etc.

The mathematical context is very complex for keyboard-mouse input. There is no intuitive way of entering mathematical expressions to a computer. There are visual interfaces like Microsoft Equation Editor, Scientific Notebook or \TeX language but they require a knowledge of their language/interface. Still with that knowledge it is not possible to input mathematical expressions as fast as doing with a pen.

Considering the developments in CPU speeds, increasing number of pen-enabled devices, ease of inputting mathematical expressions using a pen rather than keyboard-mouse, the recognition of mathematical expressions stands as a very important research area.

The mathematical expression recognition capability may also be incorporated with the existing algebra solving software, graphing programs and simulation systems to form a complete superior system which only needs a pen to interact.

1.2 Previous Work

Although the capabilities of hardware for online applications recently achieved an adequate level, the study of parsing a mathematical expression has long been studied. The very early work (1968) is done by R. H. Anderson [1] which assumed an error-free symbol recognizer and presented a coordinate grammar for 2D grammar.

Later on, Belaid and Haton [2] proposed a method based on segmentation into basic primitives, for symbol recognition. Sakamoto et al. [3] used dynamic programming for segmentation of a sequence of strokes. Chan and Yeung [4] proposed a syntactic approach which defines a set of rules for placement of symbols for parsing. After that

Zanibbi et al.[5] used a tree-transformation method for understanding 2D structure of expressions.

Symbol recognition is a subproblem aspect of a mathematical expression recognition system and several different methods have been proposed. Hidden Markov Models (HMMs) are used by Koschinski et al. [6] and Winkler et al. [7] for symbol recognition. They had 82 symbols which were written 50 times. They achieved a writer dependent accuracy of 96.9%. A combination of HMMs and neural networks is proposed by Kosmala et al. [8] . In another method proposed by Xuejun et al. [9] an improved version of Kohn-Munkres algorithm is used for symbol matching. With a 94 symbol set a writer-dependent recognition rate of 90.52% is achieved. Later on, Tapia and Rojas [10] proposed a support vector machine (SVM) based recognition with an accuracy of more than 99% in a 43 symbol set. A combination of classifiers is tested in recognition of symbols by Garain and Chaudhuri [11]. They used feature template matching together with HMMs in a 198 symbol set and achieved 92% correct classification rate.

1.3 Overview

In this study different aspects of a complete expression recognition system will be presented and further expanded into a article recognition solution. In chapter 2, an isolated symbol recognition scheme depending on a neural network is explained. In chapter 3, a method for parsing and recognizing mathematical expressions is proposed. Chapter 4 gives an overview for an article recognition system that can truly recognize a scientific article. Finally in chapter 5, the developed system throughout this study is explained and performance of the system is given.

Chapter 2

Math Symbol Recognition

The first step for building a mathematical expression recognizer is to build a recognizer for individual symbols that appear in a mathematical context.

In the next section a general overview of the complete set of symbols and previous work on recognizing individual symbols are given. Section 2 gives a brief introduction to neural networks which are used for recognition of symbols in this study. In section 3, the data collection and normalization steps are explained and in section 4, the results for symbol recognition for a single-user is given.

2.1 Math Symbol Set

The complete set of mathematical symbols is quite large. One can use a variety of character sets (Roman letters, Greek letters, operator symbols), different font styles (bold, italic, regular) and a range of font sizes (superscript, subscript, etc).

2.1.1 \LaTeX Math Symbols

\LaTeX symbol set contains several symbols that can be used in writing mathematical expressions. These symbols consist of several strokes (usually up to four) and can be written in different sizes. A comprehensive list of symbols can be found in the appendix C.

2.1.2 Selected Symbols for Online Recognition

The complete set of mathematical symbols is quite large (more than 600 symbols) compared to a character set. This makes it hard for a handwritten symbol classifier to give low-error results. A reasonable work around with the large symbol set is using a reduced symbol set which includes lower number of symbols.

In the proposed system, we are using a 66 symbol set which lets us to write trigonometric and logarithmic functions, integrals, sigma notation, fractions, some Greek letters and small letters. These symbols are shown in Table 2.1:

0	1	2	3	4	5	6	7	8	9	+
-	/	*	a	b	c	d	e	f	g	h
i	j	k	l	m	n	o	p	q	r	s
t	u	v	w	x	y	z	$\sqrt{\quad}$	Σ	cos	sin
()	[]	{	}	<	>	α	β	θ
λ	μ	=	f	∞	∂	π	tan	cot	log	ln

Table 2.1: 66 Available Symbols

2.1.3 Symbol Classification Methods

For online recognition of strokes or stroke sets different methods have been proposed in the literature.[6]

In our system, a neural network classifier is utilized for recognizing individual strokes. For the ease of segmentation, each character is assumed to be written in a single stroke. Most of the characters in the proposed set is single stroke and for multiple-stroke symbols, single-stroke equivalents are suggested. Figure 2.1 shows the whole single-stroke symbol set. The single stroke assumption of symbol structure resolves the ambiguity of which stroke belongs to which symbol and lets us easily segment intersected symbols which is not possible otherwise.

0	1	2	3	4	5	6	7	8	9	+
-	/	x	a	b	c	d	e	f	g	h
l	j	k	l	m	n	o	p	q	r	s
t	u	v	w	x	y	z	$\sqrt{\quad}$	Σ	cos	sm
()	[]	{	}	<	>	α	β	Φ
λ	μ	Σ	\int	∞	∂	∇	tan	cot	log	ln

Figure 2.1: Single Stroke Equivalents of 66 Symbols

2.2 Neural Networks

Humans can easily recognize characters, signs, distinguish a car from a building or classify similar patterns together. We can generate rules for our understanding, use these rules for identifying subjects and alter the rules when they fail to recognize or classify. The desire to understand the brain and emulate its behavior motivated people for the development of Artificial Neural Networks (ANNs).

Artificial Neural Network is a computational system that has a structure common with biological neural networks. ANNs are generalizations of mathematical models of human cognition. The first simple models for ANNs came up approximately 60 years ago, became widely used in 1950s and 1960s. It was a quiet period for ANNs around 1970s and after 1980s ANNs again became popular.

It was McCulloch and Pitts to describe an artificial neuron in 1943 [12]. They also combined neurons into neural systems for increasing the computational power. Their work defined some basic concepts of today's ANNs. The first learning scheme for an ANN is introduced by Donald Hebb [13]. His idea is further developed for allowing computer simulations to be made [14].

In 1969, it was shown that there were some very important limitations for a perceptron type of neural net for learning [15]. These limitations decreased the enthusiasm about the ANNs and little research on ANNs has been performed in the 1970s. Back propagation was invented in the 1970s but did not become widely known [17]. It was reinvented several times and became popular after 1986 [18]. In the 1980s, back propagation and Hopfield's approach [16] renewed the enthusiasm about neural networks and allowed the use of multi-layer networks. Since then ANNs are being used for clustering, classifying, approximating functions and solving constraint satisfaction problems.

Typical structure of an ANN consists of simple elements called neurons. These neurons are basic information processing units and are interconnected with certain rules defined for their connectivity. In a typical ANN, the connections multiply the output of the previous unit with a weight and serve it as an input for the next unit. Each neuron has a behavior described by its activation function and this function is usually nonlinear.

An ANN should be trained to determine the weights associated with the connections. The method for training an ANN is an important distinguishing characteristic of a neural net. The training can be categorized into two:

- **Supervised training** : This type of training uses a sequence of training vectors, each with an associated target output vector. The weights are then

calculated according to a learning algorithm. The most common methods for supervised training are Hebb rule, Delta rule, back propagation (generalized delta rule), learning vector quantization and counter propagation.

- **Unsupervised training** : This type of training groups similar input vectors together without the use of training data to state what the input is belonging to. So, input vectors are specified but there are no target vectors associated with training. The most common methods are Kohonen self-organizing maps and the adaptive resonance theory.

In this study an ANN, with supervised training, is used for classification of individual handwritten strokes representing mathematical symbols.

2.3 Data Collection

In order to train a neural network classifier for symbol classification and testing its performance, a set of samples from each symbol is needed.

For collection of handwriting samples for the 66 symbols, a Tablet PC is used together with Microsoft's Tablet-PC API. The Ink Collector class inside the Tablet-PC API handles the individual strokes, keep them in a collection and stores all the points associated with the strokes. With the help of "Sample Collection Interface" (chapter 5, section ??) several samples are collected.

From each symbol, 50 samples are collected and 40 of them are used for training and 10 are used for test. The samples are collected from only one user, so the system is tuned for one person. The performance of the system may decline if the handwriting of another person is not similar to the ones collected.

2.3.1 Pen-Input for Symbols

The Ink Collector deals with pen-down pen-up events. It stores the movement of a pen, from a pen-down position until a pen-up is reached, into a stroke object. So, a stroke consists of several points sampled from pen movement. Approximately in a second 130 points are collected by the API. Such points are shown in figure 2.2.

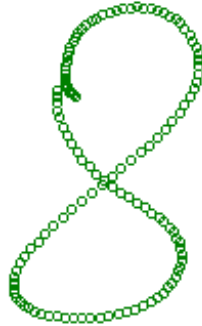


Figure 2.2: Ink Collection - Points

2.3.2 Normalization of Pen-Input

Ink collection for strokes is done in ink-coordinate system, which is automatically transferred to screen-coordinate system. But due to translation and different sizes of symbols, they are not comparable by means of a neural network. So, each symbol is translated to the origin by subtracting the means in X and Y, and they are down scaled to fit into same bounding box.

The neural network classifier needs the same number of inputs at each time for each stroke. But, the ink collection provides arbitrary number of points depending on how big the symbol is and how fast it is written. So, a subsampling is needed. In this case, a 40 input neural network is used. 20 X coordinates and 20 Y coordinates are concatenated and fed to the network. These 20 coordinates are generated by the following method:

- Separate X and Y coordinates of a stroke into two arrays
- For each array generate 20 equal-distance sampling points
- At each point calculate the value for the array by interpolation
- Concatenate those 20 X and 20 Y Values to feed to the neural network.

The output from this method is visualized in Figure 2.3:

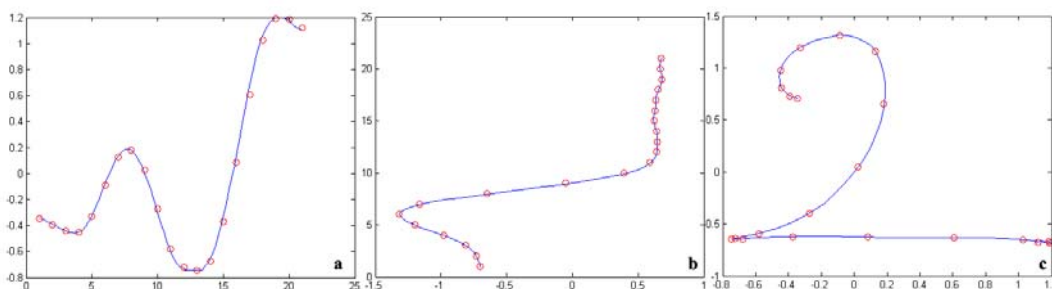


Figure 2.3: Sub-Sampling (a) X Samples, (b) Y Samples, (c) Both Samples

2.4 Classification of Individual Symbols

The symbol recognizer developed in this study can recognize 66 symbols and this implies a use of 66 output neural network. From the normalization step 40 features (20 X coordinates and 20 Y coordinates) are taken, so a 40 input neural network is needed.

A Multi-Layer Perceptron (MLP) with 40 inputs and 66 outputs is used for classification. At the input of the MLP, data is normalized by subtracting the mean and dividing by the standard deviation of the individual components of the data. So, each component of the data has zero mean and unit standard deviation. This step improves performance and is different then our previous normalization. In this step, not the individual symbols but the individual inputs of the neural network is normalized. The trained MLP contains one layer of hidden neurons. Different number of hidden-layer neurons are trained and the correct classification rate results

can be seen in Figure 2.4.

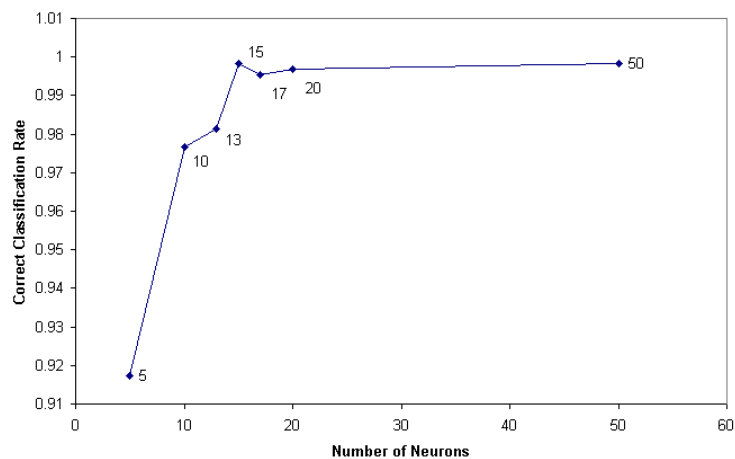


Figure 2.4: Neural Network Classifier Correct Classification Rate Graph

This figure shows that using more than 15 neurons in the hidden-layer, brings no gain in the classifier performance. During testing, at 15 Neurons, only 1 out of 641 samples is misclassified, which makes a correct-classification rate of 99.84%. Similar figures are published for single-user systems. [6, 7, 9] However, performance declines when the system is trained with samples from multiple-users as expected due to increased variability. [11]

Chapter 3

Expression Parsing

A mathematical expression recognition system should parse the written structure using the data provided from the geometric positions of the strokes and the output of the symbol recognizer.

In the next section, structure of a mathematical expression and possibilities in this study is explained. In section 2, \TeX format of writing math in a computer environment is explored. Section 3 shows how basic structures of expressions are parsed and in section 4 a recursive method for parsing multiple structures is proposed and demonstrated with an example.

3.1 Mathematical Structure

In mathematics, several notations are possible: simple algebra, matrices, calculus, theorems and proofs, etc. Thus, a good recognition system should deal with all of these structures, should be able to detect the main structure and then parse accordingly. But, current systems are able to deal with these type of structures but not possibly detect the type of main structure (especially matrices) without user help.

The proposed system in this study, forms a foundation for general expression parsing and is capable of recognizing fractions, summations, square roots, integrals, superscripts, subscripts, logarithms and trigonometric functions. It assumes that

the expression is a single mathematical statement. If there are more than one statement, it is handled by highlighting those separately from the interface (see 5.2.3).

In a mathematical context, very often basic structures are used together in a recursive manner to form more complex structures like a fraction inside an integral with square root of a number. The parser should be able to handle any number of those combinations.

3.2 Mathematical Expressions in T_EX

Although writing text with a keyboard is very convenient for any user, writing mathematical expressions and formatting them is not as easy. T_EX is a typesetting environment that allows creating mathematical expressions together with text and figures. It has some commands devoted to entry of mathematics, and contains a large set of symbols for math. By using T_EX, a mathematical context can easily be created and formatted.

T_EX uses a recursive command structure to describe the layout of mathematical expressions. This structure can handle any combination of basic structures. An example of such structure:

$$\lambda = \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{\dots}}}} \quad (3.1)$$

is generated by the code:

$$\backslash\lambda = \backslash\sqrt{1+\backslash\sqrt{1+\backslash\sqrt{1+\backslash\sqrt{\dots}}}}} \quad (3.2)$$

Also, the relative sizes and positions of the symbols define the relations. An example of such structure:

$$3^{2^1_0} \text{ or } 3_{2^1_0} \quad (3.3)$$

is generated by the code:

$$3^{\{2^{\{1^{\{0\}}}\}} \text{ or } 3_{\{2_{\{1_{\{0\}}}\}}} \quad (3.4)$$

There might be accents or dots around the symbols. This structure is hard to detect and hard to assign the relations of accents or dots to symbols correctly. An example of such structure:

$$\hat{m}, \ddot{x}, \tilde{n}, \grave{a}, \bar{X}, \vec{t}, \grave{z} \quad (3.5)$$

is generated by code:

$$\hat{m}, \ddot{x}, \tilde{n}, \grave{a}, \bar{X}, \vec{t}, \grave{z} \quad (3.6)$$

3.3 Parsing Simple Structures

Parsing a mathematical structure is quite complicated due to the unlimited possibilities of combinations of several structures which can have horizontal and vertical positional relationships. Also individual symbols can be wrongly written, and should be deleted and rewritten, which makes it hard to use the time sequence information of consecutive strokes for segmentation of structures.

Before starting parsing, all symbols are sorted from left-to-right. The proposed system does not incorporate the time relations between strokes while parsing; this way of deleting a stroke from any place and rewriting it does not change the parsing output. It starts parsing from the left-most symbol and parses to right while not all the symbols are parsed. Every symbol is parsed only once but when it will be parsed depends on where it is standing in the mathematical expression.

If a special structure is reached than corresponding routines are called for parsing that structure. These routines differ due to the type of the structure. The following subsections explains how different structures are handled.

3.3.1 Fractions

A fraction can be generated in \TeX code by writing $\text{\frac{\{\}\{\}}$. Inside the first braces the upper part of the fraction is written and inside the second the lower part.

From the point of MLP, the "-" sign and the fraction bar are the same symbol. For differentiating those and detect the presence of a fraction bar, a decision rule is defined in the grammar. In order for a symbol to be considered as a fraction bar, a minus sign should have a width more than double of the median width of all written symbols.

When a fraction bar is detected, first the upper part is parsed and then the lower part to follow up the T_EX definition of a fraction. A sample is shown in figure 3.1:

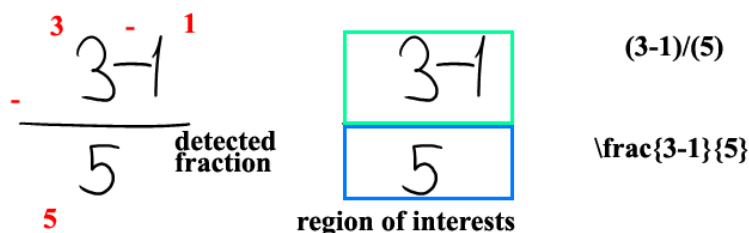


Figure 3.1: Fraction Parsing

The region of interests for fraction parsing are from the very left end to very right end of the fraction bar and from bar level to up and bar level to down. So, whatever is standing over the bar is parsed inside the first braces in T_EX code and the others are parsed inside the second braces.

3.3.2 Superscripts and Subscripts

Superscripts and subscripts are relatively small in size and stand higher or lower than the parent symbol. So, to detect those a decision rule is incorporated into the grammar. If a symbol is significantly higher and smaller than a previous symbol then it is considered as a superscript. If it is smaller but lower, than it is considered as a subscript. During testing size threshold is set to 80% and the shift from the parent characters base line is set to 35%. An example of superscripts and subscripts

is shown in figure 3.2:

Figure 3.2: Superscript and Subscript Parsing

3.3.3 Integral Parsing

Integrals can be both definitive, indefinite and can be cascaded one inside the other. A general descriptive rule for an integral defines three regions around the integral symbol: (1) lower-right region for lower limit, (2) upper-right region for upper limit, (3) right side of the integral sign for inside of the integral. These regions and a parsed integral can be seen in figure 3.3.

Figure 3.3: Integral Parsing

3.3.4 Square Root Parsing

Square-roots are defined by their enclosing rectangles. So, everything inside their bounding box is treated as it is inside the square-root. Figure 3.4 shows an example.

3.3.5 Summation Parsing

Summation in general terms have three regions: (1) lower region, (2) upper region, (3) right region. The lower and upper regions in the parsing system covers a double

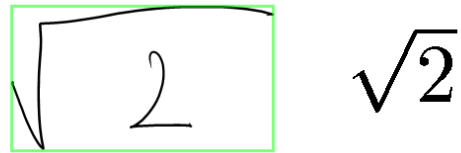


Figure 3.4: Square Root Parsing

width of the summation symbol and the right region covers whatever is on the right side. First the lower and upper regions are parsed. This way a confusion in parsing between a upper symbol appearing on the right and a symbol appearing on the right is prevented. An example is shown in figure 3.5.

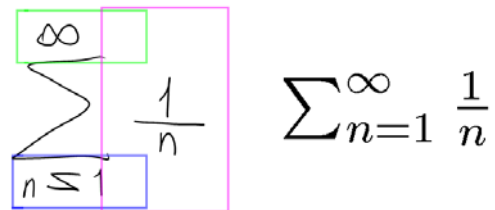


Figure 3.5: Summation Parsing

3.3.6 Other Symbols and Notations

The remaining supported notations like logarithms, trigonometric functions not require any special care so they are treated as symbols. All the remaining symbols in the expression are placed to the correct place by the parser but do not call any routine.

3.4 Multiple Structures

Mathematical expressions have a recursive structure which allows any combination of basic structures in several different relations to each other. For example a nested square-root structure may appear inside a fraction and this fraction may be part of a summation (Figure 3.6).

$$\sum_{m=1}^{10} \frac{1}{\sqrt{\sqrt{\sqrt{m}}}}$$

$$\sum_{m=1}^{10} \frac{1}{\sqrt{\sqrt{\sqrt{m}}}}$$

Figure 3.6: Multiple Structure Example

3.4.1 Recursive Parser

In order to handle these, a recursive parsing function which is capable of recognizing all the structures is needed and this type of a parser is built in this study. The parser only takes a region of interest to look for and recalls itself whenever a special structure is met with appropriate region of interest.

An empty \LaTeX string is initialized with the parser and filled while the parser is running. The parsing scheme is identical with the \LaTeX structure for writing expressions.

3.4.2 Parsing Example

An examples of the parsing process of the equation seen in figure 3.6 is explained below and illustrated in in figure 3.7. Step 1: Parser is initialized by calling the parser function by defining rectangle 1 as region of interest.

Step 2: All the symbols are sorted from left-to-right to eliminate the writing time differentiations (i.e. some symbol is deleted and written again)

Step 3: Parser starts from left-to-right. It encounters the summation sign ($\sum^{\wedge}\{\}$)

and enters the summation routine. This routine calculates rectangle 2 and calls the parser function with this rectangle. The function reads "10" and returns. (\sum^{10}_{-}) Then rectangle 3 is calculated and a parser is called with it. "m=1" is returned. ($\sum^{10}_{m=1}$). Now, the routine makes final call to the parser function to parse inside rectangle 4.

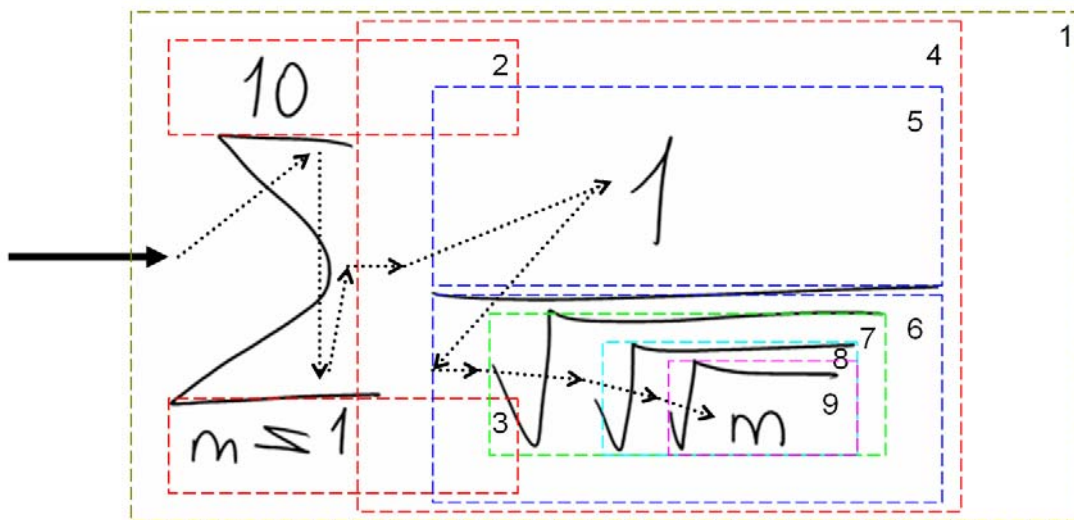


Figure 3.7: Parsing Methodology

Step 4: Inside rectangle 4, the parser reads from left-to-right and encounters a fraction. It enters the fraction routine. ($\sum^{10}_{m=1}\{\frac{\}$). This routine generates rectangles 5 and 6. First a call with rectangle 5 is done. "1" is returned. ($\sum^{10}_{m=1}\{\frac{1}{\}$). Then a call with rectangle 6 is done.

Step 5: Inside the below part of fraction (rectangle 6), parser encounters a square-root. ($\sum^{10}_{m=1}\{\frac{1}{\sqrt{\}}$). Rectangle 7 is generated. A new parser function is called. Then another square-root is seen, rectangle 8 is generated. A new call is made to the parser function and finally inside rectangle 8 another square-root is reached. Final call for parser is done with rectangle 9 and "m" is returned. ($\sum^{10}_{m=1}\{\frac{1}{\sqrt{\sqrt{\sqrt{m}}}}\}$).

Step 6: Now all the recursive calls start returning so all the remaining brackets are in place. ($\sum_{m=1}^{10} \frac{1}{\sqrt{\sqrt{\sqrt{m}}}}$). Final code can be used for generating the following equation:

$$\sum_{m=1}^{10} \frac{1}{\sqrt{\sqrt{\sqrt{m}}}} \quad (3.7)$$

Chapter 4

Article Structure Recognition

Recognition of a mathematical expression alone is not very useful; hence in our system it is possible to recognize complete articles, consisting of text, math and figures. Here, an article refers a range from a scientific paper to handwritten notes.

In the next section, a brief description of an articles' structure is given. In section 2, a methodology for building words, lines and paragraphs is explained and in section 3 a simple method is provided for identifying figures and expressions.

4.1 Article Structure

A complete system for article structure recognition should be able to manage text, mathematical expressions and figure. It should properly recognize handwritten text, expressions and combine those with the figures.

It is also very likely that inside a paragraph, text is mixed with mathematical expressions. It is very hard to discriminate those especially for handwritten documents.

Figure 4.1 simply displays all the above mentioned features. Green boxes contain text areas, orange box contains the figure and magenta boxes contain mathematical areas. Some mathematical expressions are inside in a text region (magenta boxes inside green boxes).

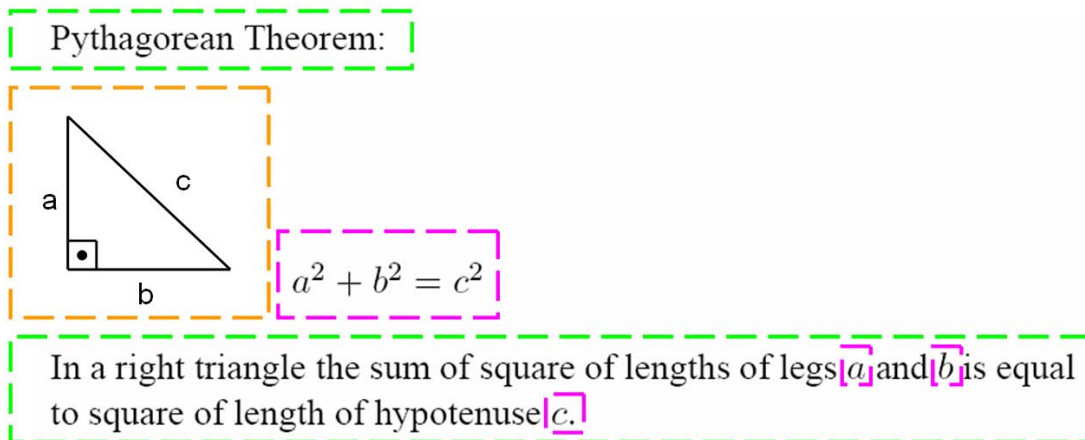


Figure 4.1: Basic Article Structure

Also it is possible to have words, lines and paragraphs in an article. An article reader system should be able to combine symbols and characters into words, words into lines and to paragraphs.

4.2 Combining Symbol & Characters

4.2.1 Word Grouping

The methodology utilized in this study is very basic and efficient. Every symbol or character is associated with a bounding box. The word combiner inflates those bounding boxes with a pre-defined value and then looks if they intersect or not. If some bounding boxes are intersecting then they are grouped into the same word.

An example of word combining is shown in figure 4.2. Each stroke has a bounding box as displayed by green boxes. Every green box is inflated and becomes a magenta box. Then a grouper looks for intersecting magenta boxes and forms the blue rectangles which are now containing words.

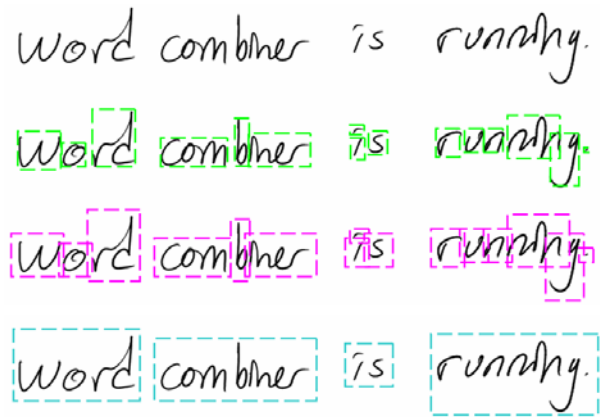


Figure 4.2: Word Grouping

4.2.2 Line Grouping

After the words are separated, the lines need to be grouped. So, each line is formed by inflating the word rectangles only horizontally.

An example of line grouping is shown in figure 4.3. Orange rectangles are formed

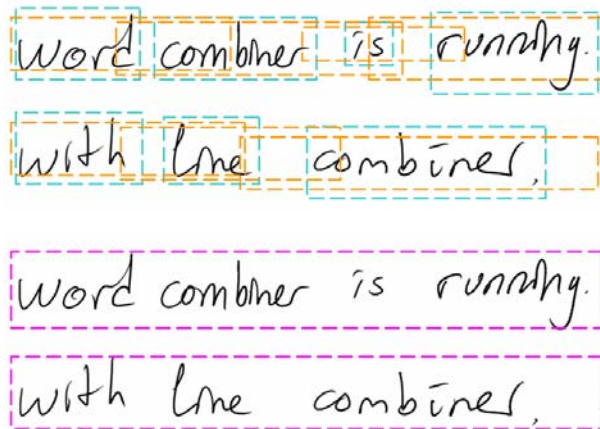


Figure 4.3: Line Grouping

by shrinking the original blue word rectangles in vertical axis and expanding heavily in horizontal axis. By shrinking in vertical axis a small line angle is tolerated and by changing the horizontal expansion the combination of farther words are guaranteed. Then the orange rectangles are searched for intersections and intersecting rectangles

grouped into lines.

4.2.3 Paragraph Grouping

After separate lines are grouped, paragraph can easily be formed by combining close lines together. This is achieved by expanding line rectangles vertically and combining intersected ones. As it can be seen in figure 4.4 close magenta lines are

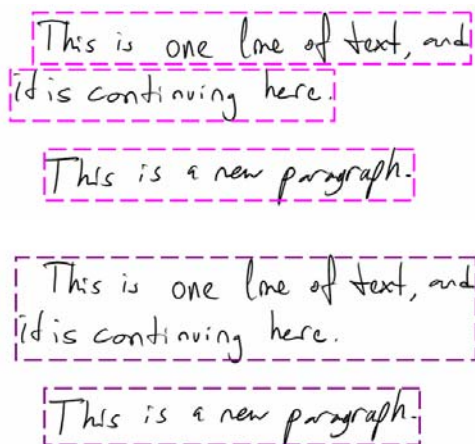


Figure 4.4: Paragraph Grouping

grouped into one paragraph and the other line is labeled as another paragraph.

4.3 Handling Figures and Expressions

Other than paragraphs and text lines, there might be figures and expressions in an article. In this study automatical detection of figures or mathematical expressions is not in our scope, but for completeness of the interface and make users easily define figures and expressions a simple method is provided as explained in the next chapter, section (5.2.3).

Chapter 5

System Specifications & User Interface

Throughout this study several interfaces are developed for collecting data and interacting with both single mathematical expressions and articles.

Next section gives an overview to the system and libraries used throughout this study. In section 2, interfaces developed in this study is explained.

5.1 General Platform Information

All the interfaces are developed in C# and built on Microsoft's Tablet PC Platform Interface. A Toshiba Tablet PC (1 GB RAM, 1.6 GHz Centrino) is utilized for both developing and testing the system. In order to classify each stroke as a symbol, MvTec's Halcon Library's built-in neural network classifier is used.

5.1.1 Microsoft Tablet PC Application Programming Interface

Tablet PC API interface supplies the necessary classes for program development on a Tablet PC. These classes provide the necessary interfaces for communicating with the pen, dealing with its output, storing, loading pen-data and recognizing the hand-written text.

The virtual ink from the pen is collected by the InkCollector class inside a col-

lection of strokes. Usually the hardware dealing with the digitization of the pen provides at least 133 samples / second at 1000 dots/inch resolution with an absolute screen-accuracy of 2 mm.

Each stroke in the inkcollector's collection stores points that are sampled while digitization, self-intersection points and cusps (peaks and radical changes of directions while drawing the strokes), etc. These data can be utilized for extracting features from strokes and hence for classification.

The Tablet PC Platform also provides a recognizer class, which is capable of recognizing handwritten text in English. So, given a set of strokes (words, lines etc.) it is possible to ask what is written. The hand-written text recognition in the interface of this study utilizes this built-in recognizer.

5.1.2 Halcon Library

Halcon is a commercial library mainly aiming applications of image processing. It also offers a powerful neural network classifier.

In our study, 50 samples are collected for each of 66 symbols which makes a total of 3300 samples. These samples are divided randomly into two parts to form train and test sets (40-10). Details of classification and performance results are given on section 2.4. Afterwards all 3300 samples are used for training the final MLP. It takes 42.7 seconds to train the final MLP on the above system.

5.1.3 Visual Studio .NET & C#

Visual Studio is a development environment which provides a set of tools to write code in several languages (C/C++, C#, Basic, J#, ASP etc.).

Due to easiness of interface development, memory management facilities and com-

patibility of Halcon library, C# is selected as the development environment.

5.2 Interfaces

With the use of described platforms and libraries 3 major interfaces are developed throughout this study. (1) Sample Collection Interface, (2) Math Only Interface, (3) Text & Figure & Math Interface.

Sample Collection Interface is used for collecting symbol samples for training the MLP classifier. Math Only Interface is capable of handling matrices, recursive structures, \LaTeX conversion and evaluation of expressions. Text & Figure & Math Interface is the complete article writing interface which is capable of segmenting article, handling recursive mathematical structures, and exporting the recognized article as PDF.

5.2.1 Sample Collection Interface

This interface is able to handle multiple users and multiple symbols. At the sample collection time the symbol to be written is displayed in top of the interface and the user is expected to write the shown symbol. At any time, the user is able to navigate back and forth between symbols and delete the symbol that is not correctly written.

This program can handle any number of users and symbols. It stores all the collected data inside an XML file. An XML file per person is kept. These files are further processed by another program which is developed for reading XML ink data, pre-process it and convert into a format which is feedable to the neural network.

5.2.2 Math Only Interface

There are two main modes while using this interface. (1) Single Expression Mode (1 by 1), (2) Matrix Mode (Anything larger than 1 by 1).

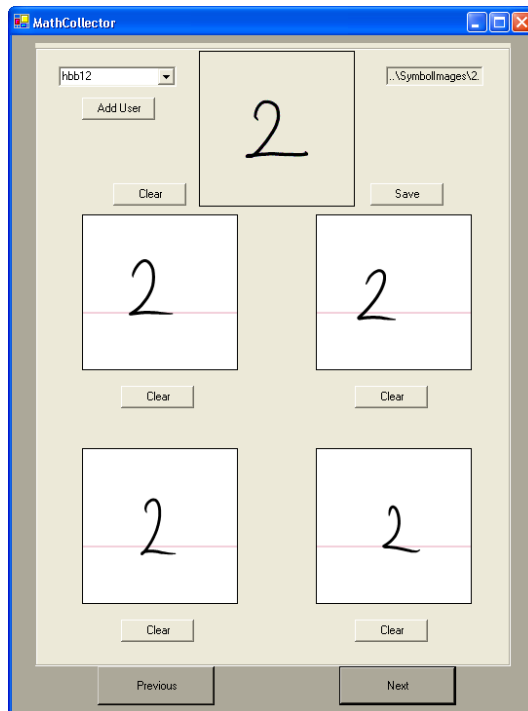


Figure 5.1: Sample Collection Interface

In both of the modes, the user is able to save, load ink data and take the corresponding \LaTeX code for the input. With the click on "LaTeX" button an easy-readable math text and a \LaTeX code is generated. The easy readable math part does not have some special \LaTeX commands and have a simpler structure then \LaTeX part. So, it is a nice place to look at for checking if something is going wrong. The LaTeX code output part displays a compilable \LaTeX code which also can be copied into any document. The "Show LaTeX" button compiles the \LaTeX code being displayed and shows the user the recognized expression.

In the single expression mode (figure 5.2), the user is able to enter a single line of math (everything is associated with one mathematical expression and parsed from left-to-right). If this single expression contains only numeric information, square-roots, powers, trigonometric and logarithmic functions than it is also evaluatable

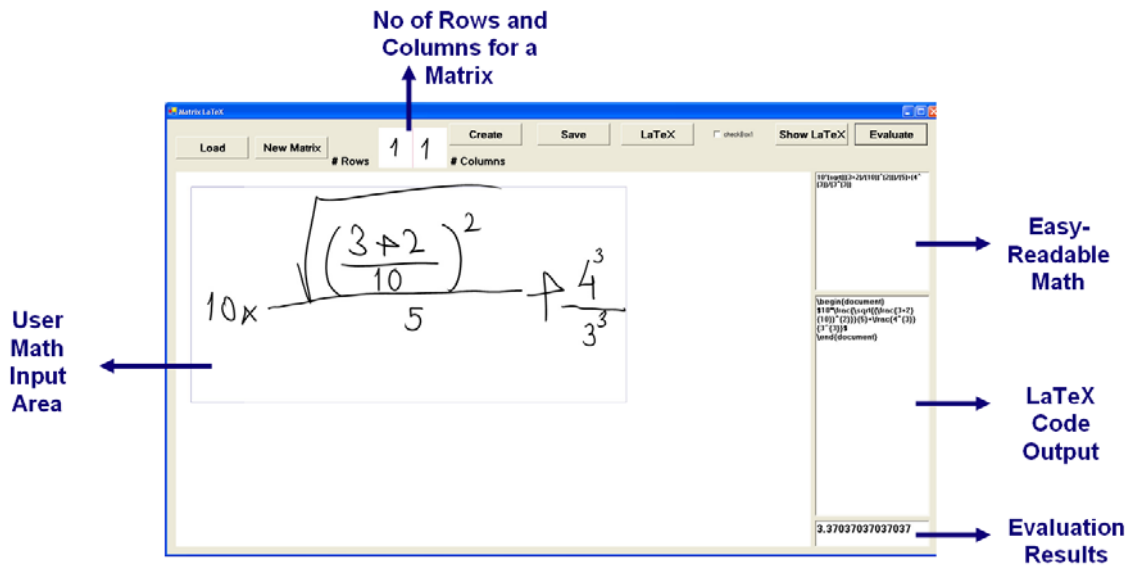


Figure 5.2: Math Only Interface

(no variables, no letters, no integral or summation notation). By clicking on "Evaluate" button, the user can easily learn the mathematical result of the expression.

In the matrix mode, the user creates an empty matrix by writing on the top boxes the desired number of rows and columns and then presses the "Create" button. Then an empty matrix with the requested number of rows and columns is displayed. Every rectangle in a matrix is a single expression, so each rectangle is parsed separately and the output is combined into an array structure while generating the \LaTeX source code. More expression examples can be seen at Appendix A.

5.2.3 Text & Figure & Math Interface

Text & Figure & Math Interface is an article recognition program which is capable of generating PDF documents from handwritten user input.

The program segments the article into words, line and paragraphs in real-time (while the user is entering data). There are three pen-types associated with the program.

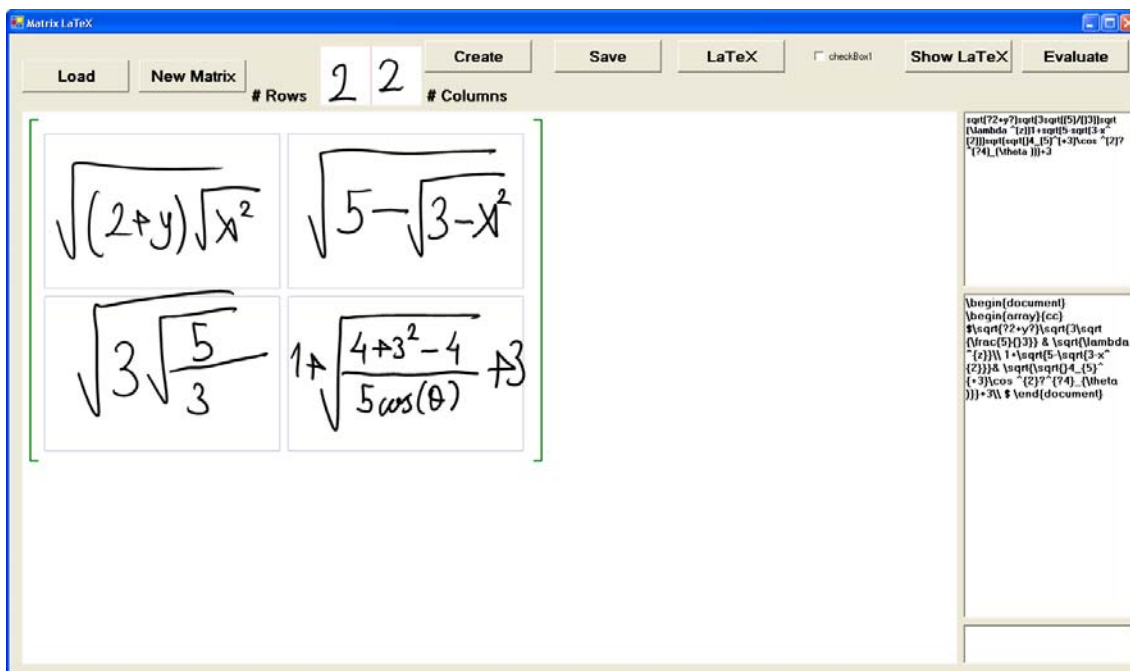


Figure 5.3: Math Only Interface - Matrix Mode

(1) Pen (Black pen for writing, (2) Highlight (To highlight Math areas), (3) Drawing (To highlight Figure areas). So, if a user wants an area to be treated as math or as drawing then those areas should be highlighted with the appropriate pen.

When the "Recognize" button is clicked, the system sorts all the ink from top-to-bottom and left-to-right, partition into paragraph, lines, words and calls the recognizer associated with the input. The normal text input is recognized by the Microsoft's built-in handwritten text recognizer, the math input is recognized by the parser proposed in this study and the figures are just converted into images and placed in an appropriate place inside the \LaTeX code. The generated \LaTeX source is displayed inside the textbox on the left side.

With the click of "LaTeX" button the recognized documents \LaTeX source is compiled into a PDF document and displayed with an external viewer. More examples can be seen at Appendix B.

Chapter 6

Conclusions & Future Work

This study explains all the aspects of an online mathematical expression recognition system and guides the building of such a system from scratch. A unique methodology, single-stroke assumption, makes the system possible to segment every symbol correctly even though they intersect. The expression parser works in parallel with the \TeX writing style for full compatibility and a recursive scheme is employed for recognition of multiple structures in a single expression.

Neural networks are efficiently used with the single-stroke symbol assumption and very-low error rate classification results are achieved for a system trained with a single-users data. A distributable interface for collecting samples is developed and used for collecting samples from a single user, but samples from multiple users have not been collected. This should be done for testing the classifier performance with multiple-users data and for having better generalizations of each symbol. Also a 66 symbol set is defined in our study and this set can be easily extended if additional samples are provided.

Equation parser handles fractions, summation notation, matrices, integrals, square roots, superscripts, subscripts, trigonometric and logarithmic functions. It should be further extended to other notations and also to special structures in \TeX like theorems and lemmas.

Since the article structure recognition is not the direct focus of this study, a simple but efficient method is utilized for recognizing parts of a written document. In order to detect the presence of an expression or a figure and to decrease the user interaction with the interface, a better algorithm should be applied.

Developed interfaces in this study comprehensively covers the possible uses of an online mathematical expression recognition system. All the interfaces are designed in such a way that they can support modifications in the basic building blocks (symbol recognizer, expression parser, article structure recognizer), hence the system created in this thesis can easily be further developed.

Appendix A

Examples of Parsed & Recognized Expressions

$$\begin{array}{l}
 -\sqrt{\frac{\sqrt{315}}{2 \times \sqrt{5} + 4}} + 3 \\
 \hline
 -4 \times 3\sqrt{5} \\
 \frac{-\sqrt{\frac{\sqrt{315}}{2 \times \sqrt{5} + 4}} + 3}{-4 \times 3\sqrt{5}}
 \end{array}$$

Figure A.1: Expression Example 1

$$\begin{array}{l}
 \frac{3^{32} + 4^{12}}{\sqrt{5^{13}}} + \frac{\sqrt{3+4^2}}{5} \\
 \frac{3^{32} + 4^{12}}{\sqrt{5^{13}}} + \frac{\sqrt{3+4^2}}{5}
 \end{array}$$

Figure A.2: Expression Example 2

$$\begin{array}{l}
 1 + \frac{2 + \frac{3 + \frac{7}{8}}{6}}{4^2 + \frac{9}{5}} \\
 1 + \frac{2 + \frac{3 + \frac{7}{8}}{6}}{4^2 + \frac{9}{5}}
 \end{array}$$

Figure A.3: Expression Example 3

$$\sqrt{\sqrt{\frac{\sqrt{3}}{5}} + \frac{2 + \sqrt{\frac{3 + \frac{7}{8}}{6}}}{\cos^2(2\alpha) + \sin^2(2\beta)}}_{4 - 3,12}$$

$$\sqrt{\sqrt{\frac{\sqrt{3}}{5}} + \frac{2 + \frac{\sqrt{3 + \frac{7}{8}}}{6}}{\cos^2(2\alpha) + \sin^2(2\beta)}}_{4 - 3,12}$$

Figure A.4: Expression Example 4

$$10,1 + \frac{2^5}{3^6 + \frac{4^7}{5^8 + \frac{6^9}{7^0}}}$$

$$10.1 + \frac{2^5}{3^6 + \frac{4^7}{5^8 + \frac{6^9}{7^0}}}$$

Figure A.5: Expression Example 5

$$\sum_{n=1}^8 \sum_{m=1}^9 3mn$$

$$\sum_{n=1}^8 \sum_{m=1}^9 3mn$$

Figure A.6: Expression Example 6

$$1^{-2} + \frac{\sum_{i=1}^9 \sum_{j=1}^i i^2 \sqrt{j}}{\sum_{j=3}^8 \sum_{l=j}^8 l^j + \frac{j^2}{\sqrt{l}}}$$

$$1^{-2} + \frac{\sum_{i=1}^9 \sum_{j=1}^i i^2 \sqrt{j}}{\sum_{j=3}^8 \sum_{i=j}^8 i^j + \frac{j^2}{\sqrt{i}}}$$

Figure A.7: Expression Example 7

$$(a_1 + a_2)^2 = a_1^2 + 2a_1a_2 + a_2^2$$

$$(a_1 + a_2)^2 = a_1^2 + 2a_1a_2 + a_2^2$$

Figure A.8: Expression Example 8

$$- \frac{-1,1 + 5^{2x}}{-\sqrt{4^3}}$$

$$= \frac{-1,1 + 5^{2x}}{-\sqrt{4^3}}$$

Figure A.9: Expression Example 9

$$\int_{-\infty}^{\infty} \frac{1}{x} dx$$

$$\int_{-\infty}^{\infty} \frac{1}{x} dx$$

Figure A.10: Expression Example 10

$$10x \sqrt{\left(\frac{3+2}{10}\right)^2} + \frac{4^3}{3^3}$$

$$10 * \frac{\sqrt{\left(\frac{3+2}{10}\right)^2}}{5} + \frac{4^3}{3^3}$$

Figure A.11: Expression Example 11

$$\frac{3^{\sqrt{3}} + 4^2}{\sqrt{5^{13}}} + \frac{\sqrt{3+4^2}}{-5}$$

$$\frac{3^{\sqrt{3}} + 4^2}{\sqrt{5^{13}}} + \frac{\sqrt{3+4^2}}{-5}$$

Figure A.12: Expression Example 12

Appendix B

Examples of Parsed & Recognized Articles

This is a *math* page
created by Math Let.

Please *highlight* anything
to be treated as *math*.

Like the following:

$$\frac{\sqrt{2}}{4\cos \alpha}$$

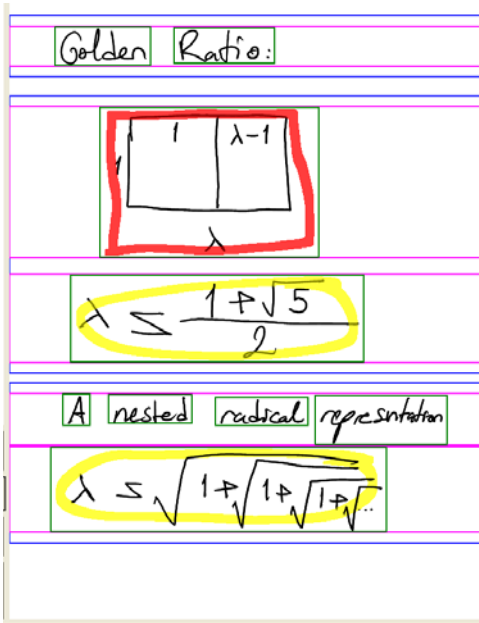
This is a *math* page by Math Let.
Please *highlight* anything to be treated as *math*.
Like the following:
 $\frac{\sqrt{2}}{4\cos \alpha}$

Figure B.1: Article Example 1



A new paragraph. The line continues if the same paragraph.
 Highlighting combines everything inside one box. Like this:
 $a + b = c$
 if they were not highlighted
 a plus b equals C
 they become separated.

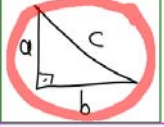
Figure B.2: Article Example 2



Golden Ratio:
 $\lambda = \frac{1 + \sqrt{5}}{2}$
 A nested radical representation $\lambda = \sqrt{1 + \sqrt{1 + \sqrt{1 + \sqrt{\dots}}}}$

Figure B.3: Article Example 3

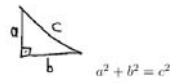
Pythagorean Theorem:



$$a^2 + b^2 = c^2$$

In a right triangle the sum of square of lengths of legs a and b is equal to square of length of hypotenuse c .

Pythagorean Theorem:



In a right triangle the sum of square of lengths of legs a and b is equal to square of length of hypotenuse c .

Figure B.4: Article Example 4

Appendix C

L^AT_EX Symbols

α	<code>\alpha</code>	θ	<code>\theta</code>	o	<code>o</code>	τ	<code>\tau</code>
β	<code>\beta</code>	ϑ	<code>\vartheta</code>	π	<code>\pi</code>	υ	<code>\upsilon</code>
γ	<code>\gamma</code>	γ	<code>\gamma</code>	ϖ	<code>\varpi</code>	ϕ	<code>\phi</code>
δ	<code>\delta</code>	κ	<code>\kappa</code>	ρ	<code>\rho</code>	φ	<code>\varphi</code>
ϵ	<code>\epsilon</code>	λ	<code>\lambda</code>	ϱ	<code>\varrho</code>	χ	<code>\chi</code>
ε	<code>\varepsilon</code>	μ	<code>\mu</code>	σ	<code>\sigma</code>	ψ	<code>\psi</code>
ζ	<code>\zeta</code>	ν	<code>\nu</code>	ς	<code>\varsigma</code>	ω	<code>\omega</code>
η	<code>\eta</code>	ξ	<code>\xi</code>				
Γ	<code>\Gamma</code>	Λ	<code>\Lambda</code>	Σ	<code>\Sigma</code>	Ψ	<code>\Psi</code>
Δ	<code>\Delta</code>	Ξ	<code>\Xi</code>	Υ	<code>\Upsilon</code>	Ω	<code>\Omega</code>
Θ	<code>\Theta</code>	Π	<code>\Pi</code>	Φ	<code>\Phi</code>		

Table C.1: Greek Letters

\pm	<code>\pm</code>	\cap	<code>\cap</code>	\diamond	<code>\diamond</code>	\oplus	<code>\oplus</code>
\mp	<code>\mp</code>	\cup	<code>\cup</code>	\triangleup	<code>\bigtriangleup</code>	\ominus	<code>\ominus</code>
\times	<code>\times</code>	\uplus	<code>\uplus</code>	\triangledown	<code>\bigtriangledown</code>	\otimes	<code>\otimes</code>
\div	<code>\div</code>	\sqcap	<code>\sqcap</code>	\triangleleft	<code>\triangleleft</code>	\oslash	<code>\oslash</code>
$*$	<code>\ast</code>	\sqcup	<code>\sqcup</code>	\triangleright	<code>\triangleright</code>	\odot	<code>\odot</code>
\star	<code>\star</code>	\vee	<code>\vee</code>	\triangleleft	<code>\lhd</code>	\bigcirc	<code>\bigcirc</code>
\circ	<code>\circ</code>	\wedge	<code>\wedge</code>	\triangleright	<code>\rhd</code>	\dagger	<code>\dagger</code>
\bullet	<code>\bullet</code>	\setminus	<code>\setminus</code>	\triangleleft	<code>\unlhd</code>	\ddagger	<code>\ddagger</code>
\cdot	<code>\cdot</code>	\wr	<code>\wr</code>	\triangleright	<code>\unrhd</code>	\amalg	<code>\amalg</code>
$+$	<code>+</code>	$-$	<code>-</code>				

Table C.2: Binary Operation Symbols

\leq	<code>\leq</code>	\geq	<code>\geq</code>	\equiv	<code>\equiv</code>	\models	<code>\models</code>
\prec	<code>\prec</code>	\succ	<code>\succ</code>	\sim	<code>\sim</code>	\perp	<code>\perp</code>
\preceq	<code>\preceq</code>	\succeq	<code>\succeq</code>	\simeq	<code>\simeq</code>	$ $	<code>\mid</code>
\ll	<code>\ll</code>	\gg	<code>\gg</code>	\asymp	<code>\asymp</code>	\parallel	<code>\parallel</code>
\subset	<code>\subset</code>	\supset	<code>\supset</code>	\approx	<code>\approx</code>	\bowtie	<code>\bowtie</code>
\subseteq	<code>\subseteq</code>	\supseteq	<code>\supseteq</code>	\cong	<code>\cong</code>	\Join	<code>\Join</code>
\sqsubset	<code>\sqsubset</code>	\sqsupset	<code>\sqsupset</code>	\neq	<code>\neq</code>	\smile	<code>\smile</code>
\sqsubseteq	<code>\sqsubseteq</code>	\sqsupseteq	<code>\sqsupseteq</code>	\doteq	<code>\doteq</code>	\frown	<code>\frown</code>
\in	<code>\in</code>	\ni	<code>\ni</code>	\propto	<code>\propto</code>	$=$	<code>=</code>
\vdash	<code>\vdash</code>	\dashv	<code>\dashv</code>	$<$	<code><</code>	$>$	<code>></code>
$:$	<code>:</code>						

Table C.3: Relation Symbols

$,$	<code>,</code>	$;$	<code>;</code>	$:$	<code>\colon</code>	\cdot	<code>\ldotp</code>	\cdot	<code>\cdot</code>
-----	----------------	-----	----------------	-----	---------------------	---------	---------------------	---------	--------------------

Table C.4: Punctuation Symbols

\leftarrow	<code>\leftarrow</code>	\longleftarrow	<code>\longleftarrow</code>	\uparrow	<code>\uparrow</code>
\Leftarrow	<code>\Leftarrow</code>	\Lleftarrow	<code>\Lleftarrow</code>	\Uparrow	<code>\Uparrow</code>
\rightarrow	<code>\rightarrow</code>	\longrightarrow	<code>\longrightarrow</code>	\downarrow	<code>\downarrow</code>
\Rightarrow	<code>\Rightarrow</code>	\Longrightarrow	<code>\Longrightarrow</code>	\Downarrow	<code>\Downarrow</code>
\leftrightarrow	<code>\leftrightarrow</code>	\longleftrightarrow	<code>\longleftrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\Leftrightarrow	<code>\Leftrightarrow</code>	\Llongleftrightarrow	<code>\Llongleftrightarrow</code>	\Updownarrow	<code>\Updownarrow</code>
\mapsto	<code>\mapsto</code>	\longmapsto	<code>\longmapsto</code>	\nearrow	<code>\nearrow</code>
\hookrightarrow	<code>\hookrightarrow</code>	\hookrightarrow	<code>\hookrightarrow</code>	\searrow	<code>\searrow</code>
\leftharpoonup	<code>\leftharpoonup</code>	\rightharpoonup	<code>\rightharpoonup</code>	\swarrow	<code>\swarrow</code>
\leftharpoondown	<code>\leftharpoondown</code>	\rightharpoondown	<code>\rightharpoondown</code>	\nwarrow	<code>\nwarrow</code>
\rightleftharpoons	<code>\rightleftharpoons</code>	\leadsto	<code>\leadsto</code>		

Table C.5: Arrow Symbols

\dots	<code>\ldots</code>	\cdots	<code>\cdots</code>	\vdots	<code>\vdots</code>	\ddots	<code>\ddots</code>
\aleph	<code>\aleph</code>	\prime	<code>\prime</code>	\forall	<code>\forall</code>	∞	<code>\infty</code>
\hbar	<code>\hbar</code>	\emptyset	<code>\emptyset</code>	\exists	<code>\exists</code>	\square	<code>\Box</code>
\imath	<code>\imath</code>	∇	<code>\nabla</code>	\neg	<code>\neg</code>	\diamond	<code>\Diamond</code>
\jmath	<code>\jmath</code>	\surd	<code>\surd</code>	\flat	<code>\flat</code>	\triangle	<code>\triangle</code>
ℓ	<code>\ell</code>	\top	<code>\top</code>	\natural	<code>\natural</code>	\clubsuit	<code>\clubsuit</code>
\wp	<code>\wp</code>	\perp	<code>\perp</code>	\sharp	<code>\sharp</code>	\diamondsuit	<code>\diamondsuit</code>
\Re	<code>\Re</code>	\parallel	<code>\parallel</code>	\backslash	<code>\backslash</code>	\heartsuit	<code>\heartsuit</code>
\Im	<code>\Im</code>	\angle	<code>\angle</code>	∂	<code>\partial</code>	\spadesuit	<code>\spadesuit</code>
\mho	<code>\mho</code>	\cdot	<code>\cdot</code>	$ $	<code> </code>		

Table C.6: Miscellaneous Symbols

Σ	<code>\sum</code>	\cap	<code>\bigcap</code>	\odot	<code>\bigodot</code>
\prod	<code>\prod</code>	\cup	<code>\bigcup</code>	\otimes	<code>\bigotimes</code>
\coprod	<code>\coprod</code>	\sqcup	<code>\bigsqcup</code>	\oplus	<code>\bigoplus</code>
\int	<code>\int</code>	\vee	<code>\bigvee</code>	\uplus	<code>\biguplus</code>
\oint	<code>\oint</code>	\wedge	<code>\bigwedge</code>		

Table C.7: Variable-sized Symbols

<code>\arccos</code>	<code>\cos</code>	<code>\csc</code>	<code>\exp</code>	<code>\ker</code>	<code>\limsup</code>	<code>\min</code>	<code>\sinh</code>
<code>\arcsin</code>	<code>\cosh</code>	<code>\deg</code>	<code>\gcd</code>	<code>\lg</code>	<code>\ln</code>	<code>\Pr</code>	<code>\sup</code>
<code>\arctan</code>	<code>\cot</code>	<code>\det</code>	<code>\hom</code>	<code>\lim</code>	<code>\log</code>	<code>\sec</code>	<code>\tan</code>
<code>\arg</code>	<code>\coth</code>	<code>\dim</code>	<code>\inf</code>	<code>\liminf</code>	<code>\max</code>	<code>\sin</code>	<code>\tanh</code>

Table C.8: Log-like Symbols

<code>(</code>	<code>(</code>	<code>)</code>	<code>)</code>	\uparrow	<code>\uparrow</code>	\Uparrow	<code>\Uparrow</code>
<code>[</code>	<code>[</code>	<code>]</code>	<code>]</code>	\downarrow	<code>\downarrow</code>	\Downarrow	<code>\Downarrow</code>
<code>{</code>	<code>\{</code>	<code>}</code>	<code>\}</code>	\updownarrow	<code>\updownarrow</code>	\Updownarrow	<code>\Updownarrow</code>
<code>⌊</code>	<code>\lfloor</code>	<code>⌋</code>	<code>\rfloor</code>	\lceil	<code>\lceil</code>	\rceil	<code>\rceil</code>
\langle	<code>\langle</code>	\rangle	<code>\rangle</code>	<code>/</code>	<code>/</code>	<code>\</code>	<code>\backslash</code>
<code> </code>	<code> </code>	<code> </code>	<code>\ </code>				

Table C.9: Delimiters

$\}$	<code>\rmoustache</code>	\int	<code>\lmoustache</code>	$\}$	<code>\rgroup</code>	$\left($	<code>\lgroup</code>
\uparrow	<code>\arrowvert</code>	\Uparrow	<code>\Arrowvert</code>	\uparrow	<code>\bracevert</code>		

Table C.10: Large Delimiters

\hat{a}	<code>\hat{a}</code>	\acute{a}	<code>\acute{a}</code>	\bar{a}	<code>\bar{a}</code>	\dot{a}	<code>\dot{a}</code>	\breve{a}	<code>\breve{a}</code>
\check{a}	<code>\check{a}</code>	\grave{a}	<code>\grave{a}</code>	\vec{a}	<code>\vec{a}</code>	\ddot{a}	<code>\ddot{a}</code>	\tilde{a}	<code>\tilde{a}</code>

Table C.11: Math mode accents

\widetilde{abc}	<code>\widetilde{abc}</code>	\widehat{abc}	<code>\widehat{abc}</code>
\overleftarrow{abc}	<code>\overleftarrow{abc}</code>	\overrightarrow{abc}	<code>\overrightarrow{abc}</code>
\overline{abc}	<code>\overline{abc}</code>	\underline{abc}	<code>\underline{abc}</code>
\overbrace{abc}	<code>\overbrace{abc}</code>	\underbrace{abc}	<code>\underbrace{abc}</code>
\sqrt{abc}	<code>\sqrt{abc}</code>	$\sqrt[n]{abc}$	<code>\sqrt[n]{abc}</code>
f'	<code>f'</code>	$\frac{abc}{xyz}$	<code>\frac{abc}{xyz}</code>

Table C.12: Some other constructions

Bibliography

- [1] R. H. Anderson, Syntax-directed recognition of hand-printed two-dimensional mathematics, Ph.D. dissertation, Dept. Eng. Appl. Phys., Harvard Univ., Cambridge, MA, 1968.
- [2] A. Belaid and J. Haton, A syntactic approach for handwritten mathematical formula recognition, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 105-111, Jan. 1984.
- [3] Y. Sakamoto, M. Xie, R. Fukuda, and M. Suzuki, On-line recognition of handwriting mathematical expression via network, in *Proc. 3rd Asian Technol. Conf. Mathematics (ATCM)*, Tsukuba, Japan, 1998, <http://www.atcminc.com/mPublications/EP/EPATCM98/>.
- [4] K.-F. Chan and D.-Y. Yeung, Recognizing on-line handwritten alphanumeric characters through flexible structural matching, *Pattern Recognit.*, vol. 32, pp. 1099-1114, 1999.
- [5] R. Zanibbi, D. Blostein, and J. R. Cordy, Recognizing mathematical expressions using tree transformation, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24, pp. 1455-1467, Nov. 2002.
- [6] M. Koschinski, H.-J.Winkler, and M. Lang, Segmentation and recognition of symbols within handwritten mathematical expressions, in *Proc.IEEE Int. Conf. Acoustics, Speech, Signal Processing (ICASSP)*, vol. 4,Detroit, MI, 1995, pp. 2439-2442.

- [7] H.-J. Winkler, H. Fahrner, and M. Lang, A soft-decision approach for structural analysis of handwritten mathematical expressions, in Proc. ICASSP, vol. 4, Detroit, MI, 1995, pp. 2459-2462.
- [8] A. Kosmala, G. Rigoll, S. Lavirotte, and L. Pottier, On-line handwritten formula recognition using hidden Markov models and context dependent graph grammars, in Proc. of Int. Conf. Document Analysis Recognition (ICDAR), Bangalore, Karnataka, India, 1999, pp. 107-110.
- [9] Z. Xuejun, L. Xinyu, Z. Shengling, P. Baochang, and Y. Tang, On-line recognition handwritten mathematical symbols, in Proc. Int. Conf. Document Analysis Recognition (ICDAR), Ulm, Germany, 1997, pp. 645-648.
- [10] E. Tapia and R. Rojas, Recognition of on-line handwritten mathematical formulas in the E-chalk system, in Proc. Int. Conf. Document Analysis Recognition (ICDAR), Edinburgh, U.K., 2003, pp. 980-984.
- [11] U. Garain and B. B. Chaudhuri, Recognition of Online Handwritten Mathematical Expressions, in Proc. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, vol. 34, No.6, 2004, pp 2366-2375
- [12] W.S. McCulloch and W. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, Bulletin of Mathematical Biophysics, 5:115-133, 1943. Reprinted in Anderson & Rosenfeld 1998, pp 18-28.
- [13] D. Hebb, The Organization of Behavior (1949), New York: John Wiley & Sons. Introduction and Chapter 4, reprinted in Anderson & Rosenfeld, 1988.
- [14] N. Rochester, H. Holland, H. Haibt, W. Duda, Tests on a Cell Assembly Theory of the Action of the Brain , Using a Large Digital Computer (1956), IRE Transactions on Information Theory, IT-2:80-93. Reprinted in Anderson & Rosenfeld, 1988.

- [15] M. Minsky, S. Papert, *Perceptrons, Expanded Edition* (1969), Cambridge, MA: MIT Press, Original Edition.
- [16] J. Hopfield, *Neural Networks and Physical Systems with Emergent Collective Computational Abilities* (1982), *Proceedings of the National Academy of Scientists*, 79:2254-2558.
- [17] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* (1974), Ph.D. Thesis, Cambridge, MA: Harvard U. Committee on Applied Mathematics.
- [18] D. Rumelhart, J. McClelland, The PDP Research Group, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition* (1986), Vol. I: Foundations, Cambridge, MA: MIT Press.