

# Mathematical expression recognition

A Degree Thesis  
Submitted to the Faculty of the  
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona  
Universitat Politècnica de Catalunya  
by  
Ignasi Mas Méndez

In partial fulfilment of the requirements for the degree in  
AUDIOVISUAL SYSTEMS FOR TELECOMMUNICATIONS  
ENGINEERING

Advisor: Antoni Gasull

Barcelona, February 2016

## Abstract

Nowadays, technology is proposing complex solutions for daily problems. These solutions include many engines. On this project, we are developing one of these engines, the Online handwritten mathematical expression recognition. Since the first OCR were proposed, many details have been studied. One of them, focused on those cases where we are trying to recognize handwritten symbols, proposed to use as input data not only an image, but the whole path that the user has written, in order to give more useful information.

In the case of mathematical expression recognition, this idea can be used to convert this data into a mathematical expression, in order to be stored or even processed (for example, to develop a handwriting calculator).

This objective is too complex to be tackled in one single project. Now we are at the beginning, so this project will focus on developing the core of this system and study its possible improvements.

## Resum

Estem en un moment en que la tecnologia proposa solucions complexes per problemes quotidians. Aquestes solucions fan ús de diversos motors. En aquest projecte un d'aquests ha estat desenvolupat, el reconeixement *Online* d'expressions matemàtiques escrites a mà.

Des del naixement dels primers OCR s'han estudiat molts detalls. Un d'ells, relacionat amb els casos on s'intenta reconèixer símbols escrits a mà, proposa en comptes d'utilitzar imatges com a informació d'entrada, utilitzar el camí recorregut per l'usuari en l'escriptura.

En el cas del reconeixement d'expressions matemàtiques, aquesta idea ens ajuda a convertir aquestes dades d'entrada en expressions matemàtiques, per ser guardades o fins i tot processades (un exemple seria el desenvolupament d'una calculadora d'escriptura a mà).

L'objectiu és massa complex per ser abordat per un sol projecte. Com estem a l'inici, aquest projecte es centrará en el desenvolupament del nucli del sistema i en l'estudi de les possibles millores.

## Resumen

Estamos en un momento en que la tecnología propone soluciones complejas para problemas cotidianos. Estas soluciones usan varios motores. En este proyecto se ha desarrollado uno de estos, el reconocimiento *Online* de expresiones matemáticas escritas a mano.

Desde que surgieron los primeros OCR se han estudiado muchos detalles. Uno de ellos, relacionado con los casos donde se pretende reconocer a símbolos escritos a mano, propone usar como información de entrada el camino recorrido por el usuario en su escritura en vez de imágenes.

En el caso del reconocimiento de expresiones matemáticas, esta idea nos permite convertir estos datos de entrada en expresiones matemáticas que pueden ser guardadas o incluso procesadas (por ejemplo si se quiere desarrollar una calculadora de escritura a mano).

El objetivo es demasiado complejo como para ser abordado en un solo proyecto. Al estar al comienzo, este proyecto se centra en el desarrollo del núcleo del sistema y en el estudio de sus posibles mejoras.

*To my parents and my sister, who are always my support.*

*To Eric, Adri, Mario, Juan and all my friends in El Pozo, who have made my time on University something so memorable.*

*To Monica, who always makes my life better.*

*This work is also yours.*

## Acknowledgements

First of all, I must state clearly that if I am able to develop this work as well as other creations, it is thanks to the people that somehow or other have made me learn the necessary amount of knowledge.

I must thank my project advisor, Antoni Gasull, for helping me to make some decisions such as focusing in one action or another during the project. I also must thank him for getting me the initial information and suggesting me about it.

I also would like to thank *CROHME* organizers for supplying me with scripts and tools to classify the training data, and also for posting such a public database, which can help many people, me included.

Finally, I would like to thank the Open-source software in general for making me avoid an unnecessary extra issue that would be having to get code and ideas from unofficial and too lax sources.

## Revision history and approval record

Revision	Date	Purpose
0	21/01/2016	Document creation
1	22/01/2016	Document revision

### DOCUMENT DISTRIBUTION LIST

Name	e-mail
Ignasi Mas	ignasi.mas.mendez@alu-etsetb.upc.edu
Antoni Gasull	antoni.gasull@upc.edu

Written by:		Reviewed and approved by:	
Date	21/01/2016	Date	22/01/2016
Name	Ignasi Mas	Name	Antoni Gasull
Position	Project Author	Position	Project Supervisor

# Index

Introduction,	11
State of the art,	19
Project development,	22
Introduction,	22
Database,	23
Segmentation,	25
Preprocessing,	28
Feature extraction,	36
Feature ponderation,	41
Template generation,	46
Classification,	51
Expression building,	56
Results,	65
Budget,	69
Conclusions and future development,	70
References,	72
Appendices,	74



## List of Figures

1	<i>Example of an application, Web equation from Vision Objects .</i>	20
2	<i>InkML specifications in W3C . . . . .</i>	21
3	<i>Results of the 2013 CROHME edition . . . . .</i>	21
4	<i>What we want from the system . . . . .</i>	22
5	<i>Those traces will be joined together . . . . .</i>	25
6	<i>Although division bar is close to =, we don't want to join them because the user has written the rest of symbols after the divi- sion bar and before the = . . . . .</i>	26
7	<i>Bounding box and center of a symbol . . . . .</i>	27
8	<i>Effect of the smoothing step on a regular symbol . . . . .</i>	29
9	<i>Effect of point clustering on a regular symbol . . . . .</i>	30
10	<i>Note the hook detection and removal of this step . . . . .</i>	30
11	<i>Effect on a regular symbol of the polygonal approximation . . .</i>	32
12	<i>Note how the point is removed on the second image . . . . .</i>	32
13	<i>A regular symbol resampled by a preselected number of points .</i>	33
14	<i>Symbol with three traces reordered, the order is green, blue, yellow . . . . .</i>	34
15	<i>Example of the distribution of a feature of two dimensions, in this case the Quadratic error for symbols with two traces, where red samples are symbol 7 and blue samples are the rest of them. . . . .</i>	41
16	<i>Some templates . . . . .</i>	46
17	<i>Different ways to draw a 7, with one or two traces . . . . .</i>	47
18	<i>Effect when two traces are joined, in this case on the symbol +. The resulting shape is wrong . . . . .</i>	48
19	<i>Distance computation between a symbol and a template (Elas- tic matching) . . . . .</i>	51
20	<i>Example of a character template (6) and the primal shapes that can build it . . . . .</i>	52
21	<i>Example of some symbols and the character they are tagged on the system . . . . .</i>	54
22	<i>Example of region delimiters for a symbol. Blue line is the bounding box, red dashed line is the outside box and black lines are the superThreshold and the subThreshold. . . . .</i>	58
23	<i>First case . . . . .</i>	60
24	<i>Duality on possible dominances . . . . .</i>	60
25	<i>Exponent added . . . . .</i>	61
26	<i>Symbol out of ranges. . . . .</i>	62

27	<i>Final result</i>	63
28	<i>Some inputs and their output when their tags are wright</i>	64
29	<i>Example of a MST</i>	64
30	<i>Wrong classification results on wrong expression</i>	65
31	<i>One bad decision is disturbing the rest of the expression</i>	65
32	<i>Intra-class variation is one of the main issues at this point</i>	66
33	<i>Wrong segmentation can ruin some cases</i>	67
34	<i>Another case of wrong segmentation effect over classification and expression building</i>	67
35	<i>Example os the system results</i>	68
36	<i>Proposed system</i>	74
37	<i>Implemented system</i>	75

## List of Tables

1	<i>Classification of characters . . . . .</i>	57
2	<i>Values for thresholds, where <math>x_s</math> and <math>y_s</math> are the right and superior limits of the bounding box and <math>W_s</math> and <math>H_s</math> the width and height of the bounding box . . . . .</i>	57
3	<i>Possible hard dominances by kind of dominant symbol. . . . .</i>	59

# 1 Introduction

Nowadays, technology is proposing complex solutions for many daily challenges. It is hard to spend a single day without using some technique developed from deep ideas. On the other hand, those technologies are usually implemented as the addition of many advances. For example, when you turn on the TV you are making use of the communication between your TV remote control and the TV itself, of the electronics inside the device, of the optical engineering applied on the screen...

On this project we are focusing on one of this daily fields. The use of calculators has been important since many years ago. It has sense that we want to improve its usage comfortability to its maximum. This project has been developed thinking on the possible use of handwriting over a touchscreen for computing operations, dessigning graphics and understand equations in general as a calculator.

It focuses on the recognition engine, a kind of OCR to recognize mathematical expressions while they are being written.

This engine can also be used for many other applications, as storing class notes or simply expressions.

## 1.1 Objectives

The main goal of this project is to dessign and implement a system to recognize a mathematical expression from handwriting data, test it and study its results, so the project can continue improving them, until they are good enough.

To test this goals I will use a database composed of many digital ink data files, and I will generate a  $\text{\LaTeX}$  expression for every input, then I will study the errors.

We don't focus neither on the front-end dessign nor on the solving task.

## 1.2 Requirements and specifications

### 1.2.1 Requirements

- Implement a system to return a  $\text{\LaTeX}$  expression from a digital ink data file
- Perform a test over a wide database
- Propose improvements to the system according to the results

### 1.2.2 Specifications

- All the system will be implemented in Python and tested on Linux
- The digital ink data format which the system will work with is *InkML*
- The database will be the one offered by the *CROHME* competition
- Many Python libraries will be used, such as *Numpy* or *Statistics*

## 1.3 Methods and procedures

This project is starting from scratch and will require deep research, so it is important to make clear the procedure to be followed.

Our first step of the project will be an information research about the topic. Information research means to find several articles and papers and read them. The study on those papers doesn't mean that we have to implement them all, but we need to have an idea of the approaches to this problem and select one of them to focus, not strictly the one that gives the best results, but the most appropriate to make the first implementation of the system. Think that the project can be continued later, so don't worry if a paper misses some good ideas, because they can be added in future improvements. Also keep those papers which give some ideas possible to be included.

Once we have selected one paper we must deepen in its ideas and make a summary/scheme of the ideal proposed system. This will probably not be our system at the end of the project, due to our own modifications or due to the lack of time.

Once we have designed it, we must look for the material we will need. We need two databases, one for training the system and one for testing it. The training database must have the digital ink data, but also the tags for the symbols on each file. For the testing database, the digital ink data is enough. Then we must begin implementing the code. We must implement it step by step, first considering our possibilities and deciding our best option, designing it, then programming and testing it and finally we must add comments to make it understandable for any possible future developer, or even for ourselves when we want to read it again.

While we are implementing the different system parts, we also must add them to a general script, in order to test the whole system while it's being implemented.

When we found a software error we must fix it immediately.

Once we have implemented all the parts we must test the whole system and make an study of the errors on the results. Then we can think about possible improvements on some blocks, that we can add and test the system again. At the end of the project, we must study the results and think about future improvements. Those ideas must be redacted on this document.

## 1.4 Work plan

### 1.4.1 Work packages

Project: <b>Method research and selection</b>	WP ref: WP1	
Major constituent: Articles	Sheet 1 of 10	
Short description: Select one method from one paper to follow during the project development	Start date:14/09/2015 End date:18/09/2015	
	Start event: Taking all the articles End event: Selecting one article	
Internal task T1: Read all the pre-selected papers Internal task T2: Choose one and make a summary/scheme	Deliverables: Paper	Dates:18/09/2015

Project: <b>Database scanning</b>	WP ref: WP2	
Major constituent: Programming	Sheet 2 of 10	
Short description: Extract the data from the acquisition files	Start date:19/09/2015 End date:25/09/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Find database Internal task T2: Program a text to data conversion	Deliverables: Program	Dates:25/09/2015

Project: <b>Symbol segmentation</b>	WP ref: WP3	
Major constituent: Programming	Sheet 3 of 10	
Short description: Group the data traces into symbols	Start date:26/09/2015 End date:9/10/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Translate coordinates into images Internal task T2: Group connected traces(after morphology) Internal task T3: Error analysis and correction	Deliverables: Program	Dates:9/10/2015

Project: <b>Symbol properties and preprocessing</b>	WP ref: WP4	
Major constituent: Programming	Sheet 4 of 10	
Short description: Fix the symbols attributes (such as bounding box, etc) and correct irregularities	Start date:12/10/2015 End date:23/10/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Symbol attributes storing Internal task T2: Preprocessing over the symbols Internal task T3: Analysis of the improvement	Deliverables: Program	Dates:23/10/2015

Project: <b>Feature extraction</b>	WP ref: WP5	
Major constituent: Programming	Sheet 5 of 10	
Short description: Store features lists for every symbol	Start date:26/10/2015 End date:30/10/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Program the storing of each symbol	Deliverables: Program	Dates:30/10/2015

Project: <b>Feature ponderation</b>	WP ref: WP6	
Major constituent: Programming	Sheet 6 of 10	
Short description: Find the optimal weights for each feature	Start date:2/11/2015 End date:15/11/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Train/test database selection Internal task T2: Template generation Internal task T3: Feature weights designing Internal task T4: Feature weights implementation	Deliverables: Program	Dates:15/11/2015

Project: <b>Classification</b>	WP ref: WP7	
Major constituent: Programming, research	Sheet 7 of 10	
Short description: Implement a classifier on the symbols and study its results	Start date:16/11/2015 End date:27/11/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Classifier selection Internal task T2: Classifier implementation Internal task T3: Classifier testing	Deliverables: Program	Dates:27/11/2015

Project: <b>Structure analysis</b>	WP ref: WP8	
Major constituent: Programming	Sheet 8 of 10	
Short description: Give structural sense to the expression	Start date:30/11/2015 End date:20/12/2015	
	Start event: Designing the block End event: Testing the program	
Internal task T1: Level/semantic sense definition Internal task T2: Procedure implementation	Deliverables: Program	Dates:20/12/2015

Project: <b>System errors fixing</b>	WP ref: WP9	
Major constituent: Programming	Sheet 9 of 10	
Short description: Modify the system to improve the results	Start date:21/12/2015 End date:08/01/2016	
	Start event: Detecting error sources End event: Testing the changes	
Internal task T1: Noisy templates detection Internal task T2: Noise solving	Deliverables: Program	Dates:08/01/2016

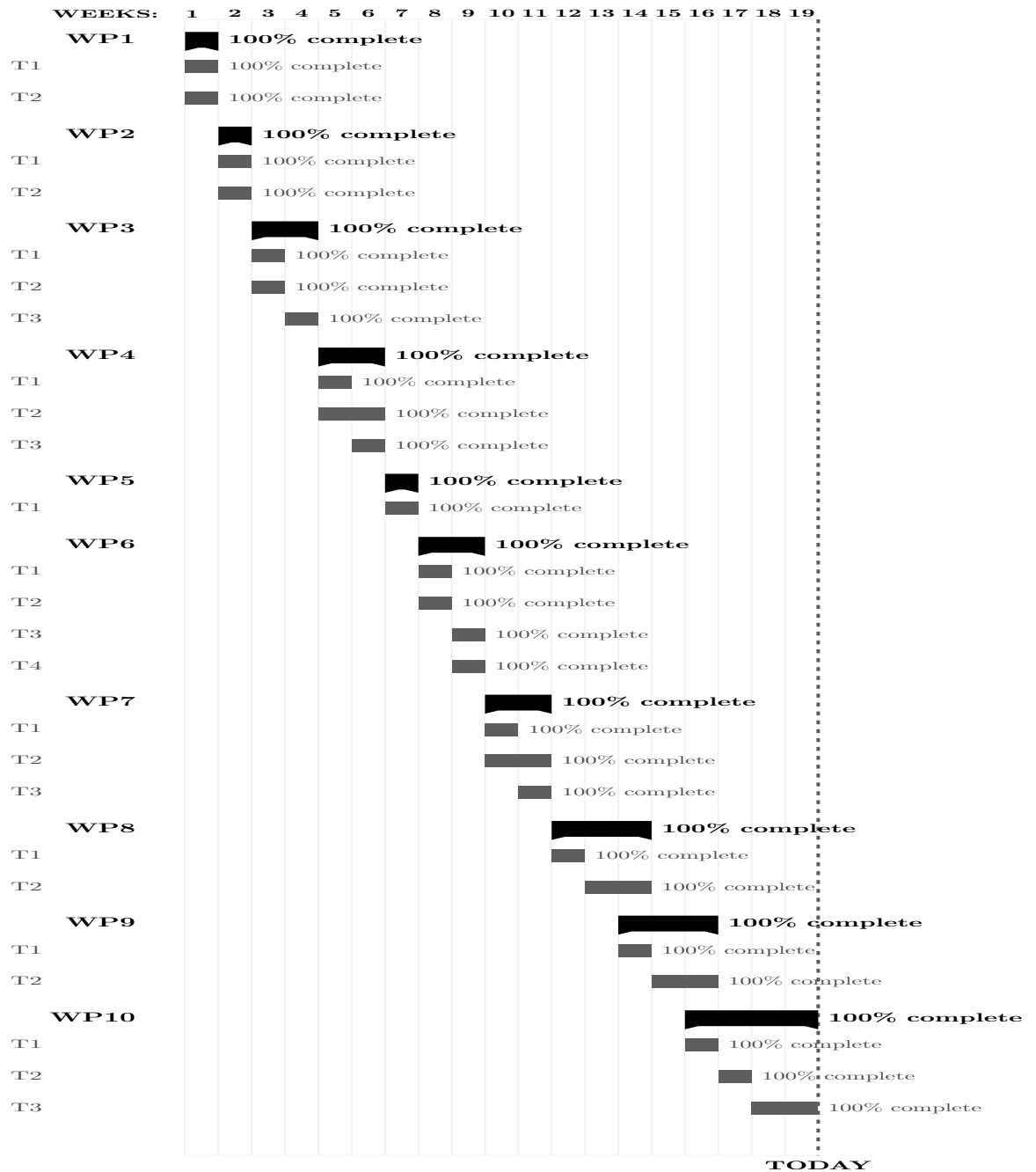
Project: <b>System testing and documentation</b>	WP ref: WP10	
Major constituent: Analysis and report	Sheet 10 of 10	
Short description: Test the system and take conclusions	Start date:09/01/2016	
	End date:25/01/2016	
	Start event: Testing the whole system End event: Report performance	
Internal task T1: Testing scripts implementation	Deliverables: Report	Dates:25/01/2016
Internal task T2: Test running and analysis		
Internal task T3: Report		



### 1.4.2 Milestones

WP	Task	Short title	Milestone	Date
WP1	T1	Read all the pre-selected papers		18/09/2015
WP1	T2	Choose one and make a summary/scheme	Summary	18/09/2015
WP2	T1	Find database	Database	20/09/2015
WP2	T2	Program a text to data conversion	Program	25/09/2015
WP3	T1	Translate coordinates into images	Program	26/10/2015
WP3	T2	Group connected traces(after morphology)	Program	6/10/2015
WP3	T3	Error analysis and correction	Program	09/10/2015
WP4	T1	Symbol attributes storing	Program	10/10/2015
WP4	T2	Preprocessing over the symbols	Program	22/10/2015
WP4	T3	Analysis of the improvement	Program	23/10/2015
WP5	T1	Program the storing of each symbol	Program	30/10/2015
WP6	T1	Train/test database selection	Program	01/11/2015
WP6	T2	Template generation	Program	08/11/2015
WP6	T3	Feature weights dessigning	Dessign	10/11/2015
WP6	T4	Feature weights implementation	Program	15/11/2015
WP7	T1	Classifier selection		17/11/2015
WP7	T2	Classifier implementation	Program	25/11/2015
WP7	T3	Classifier testing	Program	27/11/2015
WP8	T1	Level/semantic sense definition	Program	05/12/2015
WP8	T2	Procedure implementation	Program	20/12/2015
WP9	T1	Noisy templates detection	Program	28/12/2015
WP9	T2	Noise solving	Program	08/01/2016
WP10	T1	Testing scripts implementation	Program	12/01/2016
WP10	T2	Test running and analysis	Program	25/01/2016
WP10	T3	Report	Document	25/01/2016

### 1.4.3 Gantt diagram



## 1.5 Deviations and incidences

Although my original plan was to implement also the system feedback, I reorganized it and focused to achieve the best possible performance implementing the system block by block.

I also had to spend more time generating the templates, because the database only had tags about the whole expression on each file, so I also had to extract tags for each isolated symbol. Finally, the CROHME competition organizers gave me some scripts to solve it.

My original idea also included trying some proposed classifiers, but finally I implemented a variation of one of them.

Building an MST also was in my initial plan for the structural analysis, but the results suggested to improve the classification priorly than the expression building.

There has been one more incidence to mention. I would like to have worked with an InkML generator, but I did not found any available. So, I had to work with the database all the time.

## 2 State of the art

Nowadays we have plenty of tools for almost everything. Online handwritten math recognition is not different.

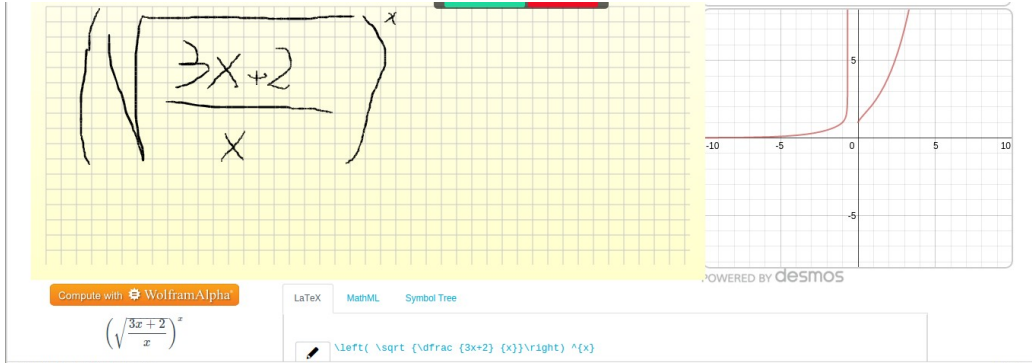
Many topics leads us to this field. First of all, more than half a century ago, handwriting recognition started to be studied for security purposes. Handwritten OCRs were quickly accepted and use in some institutions such as banks or post offices. We could talk about its evolution, but in short, they evolved until nowadays, that we have plenty of OCR software alternatives.

During this study over decades, some researchers found that capturing Online data (which means capturing the data in real time, while is being written, against the traditional method of Offline data) gave extra information that could be useful in order to get better results. Since then, much of the research has been about Online recognition.

On the other hand, mathematical expression recognition has also evolved. Decades ago, some researchers began to wonder if it could be possible to recognize handwritten mathematical expression from a static image. While OCR were evolving, this field also added the Online research, but there were many issues on mathematical expression recognition. First of all, there are less constraints on the mathematical alphabet that on any other language, in terms of writing linearity, segmentation patterns...

There has been an increase last years of the input devices which use Online data capturing (instead of the traditional keypads), including PDAs, smartphones and tablets. That's why, although as we have mentioned, Online data capturing gives less benefits for math recognition, the increase of the amount of applications to satisfy demands more handwritten math Online recognition nowadays. Applications such as handwriting calculators or class notes digitalizers are examples that need handwritten Online mathematical expression recognition on the program engine.

So we can find plenty of software services which offer this recognition (either for Windows, Unix, Android or any OS in general) such as the *Web equation* app from *VisionObjects*, or the Android app *MyScript Calculator*.



**Figure 1:** Example of an application, Web equation from Vision Objects

Normally, the general approach of the system is similar (segmentation, pre-processing, classification and structural analysis). Improving the algorithms for symbol segmentation, classification and structural analysis, as well as normalizing the users handwriting differences, are the main tasks nowadays.

## 2.1 Formats and institutions

For a generic OCR, the data would be simply an image but in our case we want something different. Online handwriting data is stored as something called *digital ink*. This concept refers to data that specifies written strokes, where each stroke specifies its captured samples. They depend on the path followed by the user, its writing speed and the sample rate of the device. We need a format able to store a list of traces, with a list of coordinates (X and Y components for each coordinate) on each trace.

The most common format for this purpose is *InkML* (Ink Markup Language), which is an *XML-based* format to describe digital ink. It published its specification on *W3C* (World Wide Web Consortium) in 2011. As it is described by their own developers, InkML was developed to make digital ink data something that can be processed for any application (such as our case). InkML stores traces composed by coordinates, metainformation (such as users ID, age, gender...) and also the tag of the expression, when necessary. Another format for storing digital ink is *ISF* (Ink Serialized Format), developed for *Microsoft TabletPC*.

Math expressions also need a format. As InkML files can include other XML languages, they normally use *MathML* (Mathematical Markup Language) to tag mathematical expressions.



## Ink Markup Language (InkML)

**Figure 2:** *InkML specifications in W3C*

Since 2011 there exists a competition of Online handwritten mathematical expression recognition, called *CROHME* (Competition on Recognition of Online Handwritten Mathematical Expressions). They have an available InkML database on their web page, and many of the mentioned softwares have participated on it.

### CROHME-III at ICDAR 2013

Here are the results from CROHME-III held at ICFHR 2013. Read the paper in [3] for more details.

Expression-level evaluation				
System	Correct(%)	<= 1 error	<= 2 errors	<= 3 errors
VisionObjects	60.36	80.33	84.95	86.14
Universitat Politècnica de València	23.40	37.85	44.71	47.84
Tokyo Univ.	19.97	34.13	40.83	42.92
Univ. Sao Paulo	9.39	18.48	24.14	27.27
Sabancı University	8.35	19.08	24.44	26.23
Czech Tech. Univ.	2.68	9.69	16.24	20.72
Univ. Nantes*	18.33	32.04	40.24	42.92
Rochester Institute of Technology (RIT)*	14.31	24.74	32.19	36.21

\* Systems from organizers and were not ranked.

**Figure 3:** *Results of the 2013 CROHME edition*

## 3 Project development

### 3.1 Introduction

Before the development itself, we need to fix what is going to be described on this report.

We want a system able to read a handwritten mathematical input and return its meaning.

To implement this, we need an input file and a system, and that will give us a  $\text{\LaTeX}$  expression.

We will use many InkML files as inputs to test the system, and the idea is that it will work for any external user entering another InkML file, which would return them a  $\text{\LaTeX}$  expression.

As the system we obviously need the code that is going to be explained at the development, but we also need a database composed by InkML files for training, what means that their symbols have to be tagged.

To implement this, I did some research, and the original idea was something like Figure 36, shown on the appendices.

But when I organized my time I realized I had no time enough for everything, so I decided to let some parts, such as the feedback, the testing for different classifiers and the support for matrices, for future development.

Then, the result at the end of this project will be something like Figure 37, also shown on the appendices.

This also will give us the same output, but obviously the results will be worse, because the system is less robust.

There is an example of what we want:



(a) *Input*

$$\frac{3\frac{1}{2}}{2}$$

(b) *Output*

**Figure 4:** *What we want from the system*

## 3.2 Database

We are at the input of the system. We must enter digital ink data, which is what will be captured on the system where the user will be working on. As we have mentioned on the State of the Art, this data is presented in *InkML* format.

We need a large database with several repetitions of at least the most used symbols. For this I used the databases given in CROHME (*Competition on Recognition of Online Handwritten Mathematical Expressions*) last competitions, concretely 2259 InkML files, where each file has a different number of symbols.

An InkML file to train has the following structure:

```
<ink xmlns="http://www.w3.org/2003/InkML">
<traceFormat>
<channel name="X" type="decimal"/>
<channel name="Y" type="decimal"/>
</traceFormat>
<annotation type="UI">2011_IVC_DEPART_F002_E013</annotation>
<annotation type="writer">depart002</annotation>
<annotation type="truth">$y_1(x) = x^2$</annotation>
<annotation type="age">26</annotation>
<annotation type="gender">M</annotation>
<annotation type="hand">L</annotation>
<annotation type="copyright">LUNAM/IRCCyN</annotation>
<annotationXML type="truth" encoding="Content-MathML">
<math xmlns='http://www.w3.org/1998/Math/MathML'>
<mrow>
<msub>
<mi xml:id="y_1">y</mi>
<mn xml:id="1_1">1</mn>
</msub>
</mrow>
...
</mrow>
</math>
</annotationXML>
<trace id="0">
10.0112 25.56, 10.0112 25.56, 10.0112 25.556, 10.0072
25.56, 10.0072 25.568,...
</trace>
<trace id="1">
10.0914 25.889, 10.0914 25.889, 10.1035 25.8931, 10.1115
25.889, 10.1356 25.885,...
</trace>
....
```



```
<traceGroup xml:id="11">
  <annotation type="truth">Segmentation</annotation>
  <traceGroup xml:id="12">
    <annotation type="truth">y</annotation>
    <traceView traceDataRef="0"/>
    <annotationXML href="y_1"/>
  </traceGroup>
  <traceGroup xml:id="13">
    <annotation type="truth">1</annotation>
    <traceView traceDataRef="1"/>
    <annotationXML href="1_1"/>
  </traceGroup>
  ....
</traceGroup>
</ink>
```

Note that there are defined traces but also trace groups, tags and more metadata.

InkML test files have different structure, because there is no need for them to have tags or trace groups. They are something like:

```
<ink xmlns="http://www.w3.org/2003/InkML">
  <annotation type="UI">2011_IVC_DEPART_F053_E040</annotation>
  <annotation type="copyright">LUNAM/IRCCyN</annotation>
  <trace id="0">
    6.51629 10.9224, 6.50425 10.9265, 6.49623 10.9586, 6.48018
    11.0027, 6.46012 11.0428,...
  </trace>
  <trace id="1">
    6.66475 11.0107, 6.66475 11.0187, 6.65673 11.0348, 6.6487
    11.0749, 6.63265 11.1391, ...
  </trace>
  ...
</ink>
```

To parse those files we must find the trace indicators and scan those parts of the code.

When training, we must store the coordinates lists specified on the files, grouped as marked and tagged as said on the InkML files. Then the system must compute them as symbols and store them on the database.

For the testing step, we only need the coordinates lists of the traces on the file, and then begin the analysis.

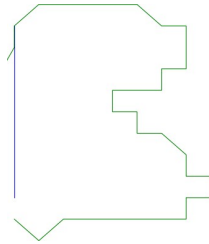
### 3.3 Segmentation

After scanning the database files, we have a set of traces, composed by their coordinates. Our goal is to group them by symbols, in order to analyze them.

#### 3.3.1 Segmentation by proximity

This approach is what I have seen more on my research. It assumes that when we read we join traces basically, if they are close enough (we also use decisively the semantical sense of the resulting symbols, but that would be done later).

We can compute this proximity by the following algorithm. As we have said, our traces are composed of coordinates. Those coordinates can be marked as white on a black background image, so we can convert the coordinates list to an image. Then, we can apply a morphological dilation to this image, in order to make the traces grow. A morphological dilation is an algorithm to expand with a selected shape, called structurant element, the white objects of an image. Then, we join traces that now are intersecting.



**Figure 5:** *Those traces will be joined together*

Our task is to find a good structural element, that is what will define this maximum distance between traces to be joined. In my case, as the structural element I used a square with a size inverse to the number of traces. Concretely, I used the following size of the square:

$$S = \sqrt{\frac{100}{N}} \quad (1)$$

where  $N$  is the number of traces and  $S$  is the size of the square.

This distance was designed empirically, testing its results.

Also, note that even being close, two traces cannot be joined together if between them the user has written traces of different symbols. An example can be seen on Figure 6.



**Figure 6:** *Although division bar is close to =, we don't want to join them because the user has written the rest of symbols after the division bar and before the =*

We can avoid joining them if we look for the intersection only for consecutive traces. For example, for each trace we can compare it to its following one, and join them if they intersect.

It is important for the system to also store which coordinates belong to each trace, as something like trace end marks, which is what I implemented.

This segmentation gives some issues, such as wrongly joined traces or segmented symbols. This is because users handwriting is not perfect, so sometimes they don't follow this distance patterns to join traces. That's why our minds cover this issue by giving a hard weight to the semantic sense on the segmentation decision.

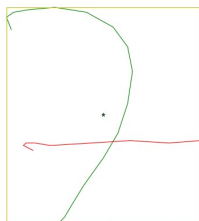
### 3.3.2 Feedback improvements

There are some ideas to emulate this semantic influence. In my research I found that HMM(*Hidden Markov Models*) can be implemented to make the system segment traces while it's classifying the symbols, what would mean to use some probabilistic method (more robust).

We also could make a set of possible segmentations depending on the selected distance and finally choose the most probable one. This can be implemented in future improvements.

### 3.3.3 Main attributes

Once we have our symbols detected, it is important to store two main attributes. For a single symbol, it's the case of its center (related with the location) and its bounding box (related with the location and size). The bounding box is between the highest and lowest coordinates on the symbol and the most at the left and the most at the right ones. The center is the point at the middle of the bounding box.



**Figure 7:** *Bounding box and center of a symbol*

## 3.4 Preprocessing

### 3.4.1 Need of preprocessing

On this step we have a set of symbols corresponding to the grouped symbols of the file. Each symbol contains a list of coordinates (for the whole symbol) and somehow and indicators for which trace does every point on the coordinate list belong to. Those parameters depend on two facts: what has the user written and how has the device captured and sampled it.

**About the user** , it's possible that two different writers (or the same at different times) write the same symbol, with the same shape, in different order. For example, someone can write an 'X' as a '\' first and then a '/' , but it can also be written in reverse order. And in the same trace, there also isn't order limitation. For example, writing a '-' left to right or right to left means the same.

**About the device** , the coordinates recorded are hardware-dependent. The same coordinates path can be written or captured faster or slower, or simply sampled different.

Also, there can be noisy information when writing a symbol. Millimetric variations from the idea due to human handwriting imperfection give useless shapes that bother more than help.

Those variations will give wrong results, because as we will compare shapes from those coordinates lists, the system will take equal shapes as completely different symbols (point by point).

That's why we need to delete unwanted irregularities and define the symbols in the same way, close to user and device independence.

### 3.4.2 Preprocessing techniques

The processes for solving those needs are called *noise reduction* and *normalization*.

**Noise reduction** is the process of deleting irregularities and useless elements.

**Normalization** is the process of redefining the coordinates list in a preselected trace and direction order and with the same number of points, to compare equivalently located points between symbols.

In this system I have implemented the following techniques to complete both processes:

### 3.4.2.1 Smoothing

It simply consists on substitute each point by the ponderation of its neighbour points. This way it can smooth the deviation from a line.



(a) *Before smoothing*



(b) *After smoothing*

**Figure 8:** *Effect of the smoothing step on a regular symbol*

We have to compute the new set of points  $p_i^*$  as this ponderation from the current points  $p_i$  and its  $n$  neighbours.

$$p_i^* = \sum_{k=-n}^n \alpha_k p_{i+k} \quad (2)$$

where

$$\sum_{k=-n}^n \alpha_k = 1 \quad (3)$$

Usually this ponderation is computed between the previous and the following point. I have used  $\alpha_{-1} = 1/4$ ,  $\alpha_0 = 1/2$  and  $\alpha_1 = 1/4$ .

This leads us to the question: what happens with the first and the last points of the list? We can't apply the same ponderation because we need to satisfy the formula in (2), so with the point itself and the following/previous one, we need to apply a ponderation that satisfies this formula. I have applied  $\alpha_0 = 2/3$  and  $\alpha_1 = 1/3$  or  $\alpha_{-1} = 1/3$  depending on if we are talking about the first or the last point.

But, is this applied only to the first and last points of the symbol? Think about the last and first points of the different traces. It has no sense, for example, to ponderate the first point of the '/' trace in 'X' with the last point of the '\' trace. So, we must apply the same ponderation than in the first and last points of the symbol in the points which are a trace end or beginning. So smoothing is applied to each trace.

The effect can be seen in Figure 8.

### 3.4.2.2 Point clustering

This step also ponderates each point with its neighbours, but this time it considers this neighborhood as an area(with a radius) instead of a set of

consecutive points. It also smooths the shape, removing small local variations considered noise.

$$p_i^* = \frac{1}{\#(V_r(p_i))} \sum_{p_k \in V_r(p_i)} p_k \quad (4)$$

where  $V_r(p_i)$  means the area of radius  $r$  around the point  $p_i$ .

I used a radius  $r = L/80$  where  $L$  is the total length of the trace, so it will consider a bigger area for long traces.

For each point it will compare the distance between every one on the same trace and the threshold radius. If it's closer, the system will include it to the cluster. Note that we don't have to compare points of different traces, because it has no sense to cluster between different segments.

This time, as it will compute the neighborhood as an area, it will consider all points in a trace equally.



**Figure 9:** *Effect of point clustering on a regular symbol*

### 3.4.2.3 Dehooking

It consists on removing 'hooks' from the trace extremes. Hooks are particular shapes that consist on closed angles near the first or the last point. They don't give any information and are considered noise.



**Figure 10:** *Note the hook detection and removal of this step*

The system searches for hooks on the symbol, and if it detects one, ends or begins (depending on the extreme) the trace on the point where its closed angle is located.

A point is considered to own a hook if two conditions are satisfied:

**First one:**

$$\theta_i < \theta \quad (5)$$

where  $\theta_i$  is the angle located at the point  $i$  (between  $\overline{p_{i-1}p_i}$  and  $\overline{p_i p_{i+1}}$ ) and  $\theta$  is a threshold angle (I have used  $17/36\pi$ ).

**Second one:**

$$\sum_{k=1}^{i-1} \|p_{k+1} - p_k\| < \alpha L \quad (6)$$

or

$$\sum_{k=i}^{n-1} \|p_{k+1} - p_k\| < \alpha L \quad (7)$$

where  $L$  is the trace length and  $\alpha$  is an adjustable parameter (I have used 0.12).

For each point the system looks at the segment formed with the previous and the next ones. This way it can compute the angle on the point, and if it detects a closed angle it asks if it's happening on a trace extreme. If it is, deletes the following points (if it's on the end) or the previous ones (if it's on the beginning).

One possible conflict is that there are some equal consecutive points. If this happens it can be an issue to take the segment between both to compute the angle, because it doesn't exist. For this, I have made my program look if it's on this case, and if it is, look for the consecutive points, until it finds a different one. Then it computes the segment with this different point.

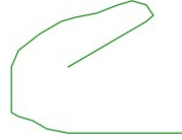
#### 3.4.2.4 Polygonal approximation

As its name tells, this step consists on representing a shape simpler, approximating it to a polygon. It is also computed trace by trace.

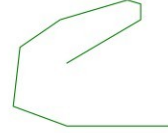
It takes at first both extremes  $A$  and  $B$ , and draws the segment  $\overline{AB}$ . Then it searches the point on the trace with the maximum euclidean distance to the segment. We can call it  $C$ . Then we have the shape approximated with two segments  $\overline{AC}$  and  $\overline{CB}$ . We can repeat the same step with both segments. The system repeats the step until a level of *tolerance* is reached, which is the maximum position difference between the original and the approximated shapes. We can see an example on the following figure.

I have used a tolerance relative with the symbol size,  $d/25$  where  $d$  is the diagonal of the symbol bounding box.





(a) *Before polygonal approximation*



(b) *After polygonal approximation*

**Figure 11:** *Effect on a regular symbol of the polygonal approximation*

### 3.4.2.5 Point deleting

In fact, this is the first technique to use for the reason I'm going to explain, but I'm describing it now for showing you its transcendence.

I'm introducing the case which consists on a trace containing one single point. Note that we can't apply any of the previous techniques, giving many issues when computing them on it. That's why it must be the first step.

Also think that we can consider an isolated point as noisy information. If not, we obviously couldn't remove it.

But note that there is one controversial case. It is when we have a list of more than one coordinate, but they are all the same. We must also consider it an isolated point, for the same reasons.

I realized of this issue empirically, while computing.



(a) *Symbol with points*



(b) *Points deleted*

**Figure 12:** *Note how the point is removed on the second image*

### 3.4.2.6 Arc Length resampling

This is the first normalization step. It consists on redefining the symbol shape with a concrete number of points, keeping the proportional length of every trace. This will let us compare symbols point by point. In my system I normalized it with 50 points on the symbol.

Note that the number of points is not the same than the number of segments. In fact, the number of segments is the number of points minus the number

of traces, because we have to discount one segment for each trace (the trace extremes are not united, so this segment is missing).



**Figure 13:** A regular symbol resampled by a preselected number of points

What the system does is, for each trace, to compute the location of the wanted points. Then for each current segment on the trace it looks if there should be new points, and if it's the case it stores them.

#### 3.4.2.7 Stroke direction and order

Once we have a list of coordinates with the same length for all symbols, we must assure that we are referring to equivalent points for each symbol. That means that we must normalize the coordinates list order.

Note what we have: a set of traces containing a set of points each one. So, we must order equally the coordinates inside each trace and also equally the traces on the symbol.

**About the points** , we must decide an standard order. I have used left to right, up to down. Now, we must compare the extremes of the trace depending on the type of symbol

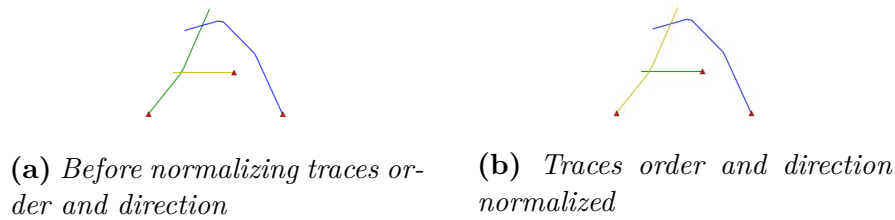
Symbols and traces are classified as *horizontal*, *vertical*, *diagonal* and *closed*. To understand those classes we must define the distance between extremes,  $R$ , as a composition of  $R_x$  and  $R_y$  (horizontal and vertical distances), and a  $\delta$  threshold (I have used a value of  $\delta = 0.5$ ).

- Horizontal if  $R_x > \delta$  and  $R_y < \delta$
- Vertical if  $R_x < \delta$  and  $R_y > \delta$
- Diagonal if  $R_x > \delta$  and  $R_y > \delta$
- Closed if  $R_x < \delta$  and  $R_y < \delta$

This classification is stored for whole symbols, but it's also computed for traces in the same way. This is what will let us standardize the traces direction.

For horizontal traces, if the first point is more on the right than the last one, it inverts the coordinates list on the trace. For vertical and diagonal traces it does the same process if it finds the first point lower than the last one. Finally this is not computed for the closed ones.

**About the traces** , we also must decide an standard order. In this case, we order it by angle of the last extreme of the trace with the horizontal direction, from low to high angles.



**Figure 14:** *Symbol with three traces reordered, the order is green, blue, yellow*

Note that if we reorder the coordinates list, we also have to update the trace indicators, because now the points of a trace will be located on another part of the list.

#### 3.4.2.8 Stroke reduction

This step consist on group strokes when there are more than a wanted number. This is rarely used, but in my system it limits the traces to 3.

I also implemented the option to add strokes if there are less than your wanted number, it simply divides the last points on different traces.

#### 3.4.2.9 Size normalization

We have normalized the number of points and its order, so now we can compare equivalent points between different shapes. The only thing to improve is the sense of the values of this coordinates. To compare shapes independently of its scale, which depends on the device, user, and other facts, we must normalize its size. I normalized it between  $-1$  and  $1$  for both horizontal and vertical axis.

So, the method is as easy as moving the symbol to our wanted dominion.

Be careful in this step, because if we are analyzing the file, we must keep the information about its original location and size. So, we must keep the original bounding box and center although we normalize the coordinates.

### 3.4.3 Steps

In my system I have used those techniques in the following order:

- Point deleting
- Smoothing
- Point clustering
- Dehooking
- Polygonal approximation
- Point deleting again
- Arc length resampling
- Stroke direction and order
- Stroke reduction
- Size normalization

## 3.5 Feature extraction

At this point we have the coordinates list for each symbol that we will use to classify them. From this raw data we must compute some parameters that we will directly compare between the untagged symbols and the templates on the database.

Taking useful features will be a key step to obtain good results. Useful features are those which discriminate a lot between symbols. For example, the first coordinate value is not a good feature, because it doesn't matter for any symbol where it begins while any different symbols can be written starting from the same point.

Remember that even selecting an appropriate set of features, there will be some better than others. That's why after the feature extraction we have to apply also a feature ponderation.

I used the features explained in [8], which I will explain more deeply later. Basically, I used features of two kinds:

**Local features** are those which are associated to every point of the coordinates list, giving information about them as well as their relation with its neighbours, the trace they belong to or simply the whole symbol. Obviously they are vectors of single elements (or tuples when they give values for X and Y components) with the same length than the coordinates list.

**Global features** are those which are associated to the symbol or to its traces. They are single values or tuples when they give information about the whole symbol, or vectors of them when referring to each trace.

### 3.5.1 Local features

#### 3.5.1.1 Coordinates

Coordinates by themselves are the first feature to extract. We don't need to compute anything to obtain them, because on this point we already have them, but we need to store them and use them later to compare like any other feature.

$$(x_i, y_i) \text{ for } p_i \text{ in symbol coordinates} \quad (8)$$

This feature is very important because it defines the traces path, and it's specially sensitive to normalization. Think about an *A* where the bar in the middle - is written left to right, and think also with another one where at contrary, this bar is written right to left. They may have exactly the same

shape but when comparing their coordinates, we could be comparing totally different points.

### 3.5.1.2 Turning Angle

This feature means the direction of the segment written from every point until the following one. It computes the angle of the horizontal direction and this segment and stores this value.

$$\theta_i = \arccos \frac{x_{i+1} - x_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \text{ for } p_i \text{ in symbol coordinates} \quad (9)$$

or

$$\theta_i = \arcsin \frac{y_{i+1} - y_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \text{ for } p_i \text{ in symbol coordinates} \quad (10)$$

It gives an idea of the patterns of his writing path. For example, if a symbol begins with a line going up it will store this pattern.

For the last point of each trace I always gave the  $\pi$  value.

### 3.5.1.3 Turning Angle Difference

This feature is related to the Turning Angle. It's simply its difference on the studied point and the previous one. This means that it's giving the angle of the incident segments to the point.

$$\Delta\theta_i = \theta_i - \theta_{i-1} \text{ for } p_i \text{ in symbol coordinates} \quad (11)$$

It refers to the symbol shape, independently from its orientation (it's computed only with relations, not with absolute values). That's the main difference with the previous feature.

For the first and last points of each trace I gave the value of 0.

### 3.5.1.4 Length Position

As every symbol has a line length, this feature consists on computing the line length until each point in relation to the whole length, normalized to 1.

$$L_i = \frac{\sum_{k=1}^{i-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2}}{L} \text{ for } p_i \text{ in symbol coordinates} \quad (12)$$

where  $L$  is the total line length:

$$L = \sum_{k=1}^{N-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2} \quad (13)$$

Length Position gives an idea of where are longest segments located on the symbol writing path.

Obviously, the first value will be 0 and the last one will be 1. The rest of them will be between those values.

### 3.5.2 Global features

#### 3.5.2.1 Center of gravity

The first global feature we must implement is computed for each trace in the symbol. It is found as the average of the point coordinates in a stroke.

$$CoG(j) = \left( \sum_{i=m}^n \frac{x_i}{n-m}, \sum_{i=m}^n \frac{y_i}{n-m} \right) \text{ for } m \dots n \in \text{stroke } s_j \quad (14)$$

The meaning of this is to locate the main position of every trace, so we will compare them to find the template with the most similar trace positions.

#### 3.5.2.2 Length in Stroke

As its name suggests, it simply consists on storing the length of each trace. So we will get if a symbol is composed by long/short lines.

$$LiS(j) = \sum_{k=m}^{n-1} \sqrt{(x_{k+1} - x_k)^2 + (y_{k+1} - y_k)^2} \text{ for } m \dots n \in \text{stroke } s_j \quad (15)$$

#### 3.5.2.3 Relative Stroke Length

This feature is computed for each trace as the distance between both of its extremes in relation with the stroke length.

$$\frac{d(j)}{LiS(j)} \quad (16)$$

where

$$d(j) = \sqrt{(x_n - x_m)^2 + (y_n - y_m)^2} \text{ for } m \dots n \in \text{stroke } s_j \quad (17)$$

It gives an idea of how closed is a stroke. If its extremes are far from each other in relation with the stroke length, it means that this trace is clearly

open, similar to a line. At the contrary, long strokes with close extremes are very closed.

If the Relative Stroke Length gives a value of 1, it means that the stroke is a segment. If this value is 0, we are talking about a totally closed shape. So, with this feature and the previous one we'll know the position and the similarity to a segment against a closed shape for the traces in a symbol.

#### 3.5.2.4 Accumulated Angle

This feature is related with the segment directions again. It refers to the sum of angles in a stroke.

$$\theta_a(j) = \frac{1}{2\pi} \sum_{i=m}^n \theta_i \text{ for } m \dots n \in \text{stroke } s_j \quad (18)$$

This also gives an idea about how closed is the trace, in an angular scale.

#### 3.5.2.5 Quadratic error

It means the deviation (in sense of quadratic error) from the segment between both of the extremes of a trace to its points. It's computed as the average of this distance for the points belonging to each trace.

$$E(j) = \frac{1}{n-m} \sum_{i=m}^n d_i^2 \text{ for } m \dots n \in \text{stroke } s_j \quad (19)$$

It's also useful to know how closed is a trace, in a sense of covered area. Segments will have low quadratic errors while closed shapes will have high ones.

#### 3.5.2.6 Style

We have already talked about this classification in the preprocessing part, and I included it as a feature.

To be compared it needs a numerical value. That's why I gave it this values:

**Diagonal symbols** have the value of [1, 1]

**Horizontal symbols** have the value of [1, 2]

**Vertical symbols** have the value of [2, 1]

**Closed symbols** have the value of [2, 2]



### 3.5.3 Feature information

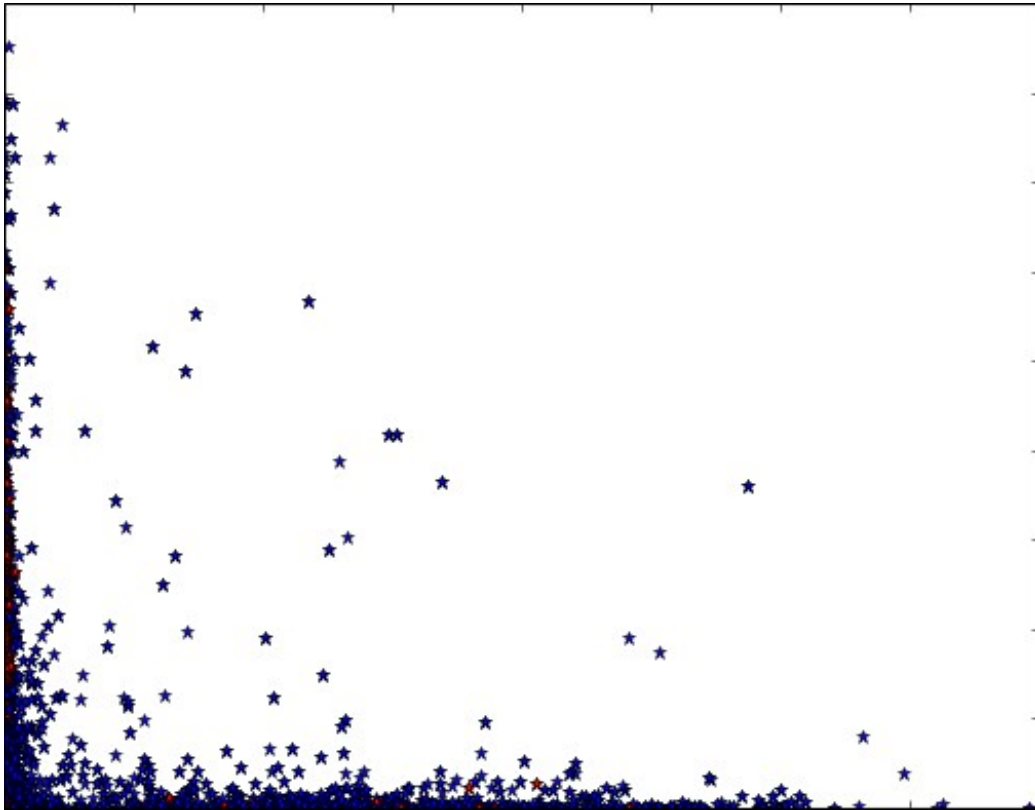
So, after computing all this features we will obtain the following data about the  $N$  sampled points distributed in  $T$  traces:

- The point distribution, described by an  $N \times 2$  vector (Coordinates)
- The point orientation, described by a vector of  $N$  (Turning Angle)
- The shape independently from the symbol orientation, described by a vector of  $N$  (Turning Angle Difference)
- The line length evolution, described by a vector of  $N$  (Length Position)
- The ubication of the traces, described by a vector of  $T \times 2$  (Center of Gravity)
- The total perimeter of the line in a symbol, described by a vector of  $T$  (Length in Stroke)
- The shape of traces, in a sense of angles(described by a vector of  $T$ , Accumulated Angle), in a sense of written perimeter of the shape(described by a vector of  $T$ , Relative Stroke Length) and in a sense of covered area (described by a vector of  $T$ , Quadratic Error)

### 3.6 Feature ponderation

Once we have completed the feature extraction we have data ready to be compared. We can compare equivalent values between the currently studied symbol and the templates on the database to find the most appropriate tag. There is a necessary step before this comparison. Think about what has been explained before. The selection of good features is essential to obtain good results. But are they all at the same level? It seems obvious that, if you have selected a large set of features where one of them is better than the rest, it should be more decisive in the final decision.

That's why we must add some weightings to the features. The objective now is to find which are the best ones.



**Figure 15:** *Example of the distribution of a feature of two dimensions, in this case the Quadratic error for symbols with two traces, where red samples are symbol 7 and blue samples are the rest of them.*

### 3.6.1 Feature concentration computation

By definition, to find a good feature we have to maximize the intraclass variation while minimizing the interclass variation. This way, in an extensive feature distribution, we can locate each symbols zone with as few false positives or false negatives as possible.

This variation can be computed by many parameters. I used standard deviation, but I also could have used the variance, for example. Remember that the standard deviation is calculated as:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2} \quad (20)$$

where  $x_i$  means the  $i$ th given sample,  $N$  means the number of samples and  $\mu$  is the mean:

$$\mu = \frac{\sum_{i=1}^N x_i}{N} \quad (21)$$

So, we know what to find with this values for each feature. Given a class  $C$ , we want:

$\downarrow \sigma_C$  where  $\sigma_C$  is the standard deviation within the class

and

$\uparrow \sigma_T$  where  $\sigma_T$  is the standard deviation for the whole set of samples.

Given both conditions, we can conclude that we want to minimize:

$$\downarrow \frac{\sigma_C}{\sigma_T} \quad (22)$$

This is what I computed to obtain the concentration of each feature. In my case, more concretely I computed  $\sigma_C$  as the sum of all classes, what means of all symbols:

$$\downarrow \frac{\sum_{\forall C \in dB} \sigma_C}{\sigma_T} \quad (23)$$

Note that each feature has its own scale. That's why in a feature with a large scale it's easier to obtain high standard deviation values, even if it's very concentrated. For example, if the values are 1010 and 990, the standard deviation is 10 while if they are 18 and 2 it will be 8. The first case has a higher standard deviation but it clearly seems more concentrated than the

second one. For this reason normalizing with the feature scale for all its values on the database seems more fair.

But if we look at the equation we'll see that it's not necessary, because we are computing a relation:

$$\downarrow \frac{\sum_{\forall C \in dB} \frac{\sigma_C}{\mu_T}}{\frac{\sigma_T}{\mu_T}} = \frac{\sum_{\forall C \in dB} \sigma_C}{\frac{\mu_T}{\mu_T}} = \frac{\sum_{\forall C \in dB} \sigma_C}{\sigma_T} \quad (24)$$

The last fact we have to consider is that not all of our features are single values (in fact, any of them). Calculating the standard deviation of a feature depends of the number of dimensions it has.

### 3.6.1.1 Tuples: $X$ and $Y$ components

When we have a feature that consists on a pair of values, for both  $X$  and  $Y$  components, we can take it like in the quadratic error:

$$\sigma_f = \sqrt{\sigma_{f_x}^2 + \sigma_{f_y}^2} \quad (25)$$

where  $f$  means the feature.

That's the case of the Style (with the numerical value previously given).

### 3.6.1.2 Vectors with each point

If we are studying a feature with one value for each point on the symbol, we basically can compute its standard deviation as the mean of them for each point. This way:

$$\sigma_f = \frac{\sum_{i=0}^N \sigma_f(i)}{N} \quad (26)$$

where  $N$  is the number of points in the feature.

But if we look at the final equation, as we know that we have normalized all the symbols to have the same number  $N$  of points, we'll see that we can remove the denominator:

$$\downarrow \frac{\sum_{i=0}^N \sigma_{fC}(i)}{\frac{N}{N}} = \frac{\sum_{i=0}^N \sigma_{fC}(i)}{\sum_{i=0}^N \sigma_{fT}(i)} \quad (27)$$

That's the case of the Turning Angle, the Turning Angle Difference and the Length Position.

### 3.6.1.3 Vectors with two values for each point

If the feature is composed by an  $X$  and an  $Y$  component for each point, we can apply both previous steps. Then we can compute a general standard deviation as the mean(for each point) of the quadratic calculation(for its  $X$  and  $Y$  components) of its values.

It's the case of the Coordinates.

### 3.6.1.4 Vectors with each stroke

In this case, we can apply the same reasoning we applied to the vectors with each point, but here we have a main difference. While in the other case we had the same number of values for this feature in all the symbols, now we can vary this number. So, we'll compute the mean again:

$$\sigma_f = \frac{\sum_{i=0}^S \sigma_f(i)}{S} \quad (28)$$

where  $S$  means the number of strokes.

Our problem here is that we can't compute a generic total standard deviation, because in some cases we have to divide by one, in some others by two...Instead, we can separate our three cases on the database (one stroke, two strokes and three strokes), sum them and divide them by 3, to compute a kind of mean. Or more precisely, ponderated by their frequency of the database(to compute the mean is more fair to weight more the cases with more repetitions on the database). Mathematically we can express:

$$\begin{aligned} & \downarrow \frac{\sum_{j=1}^{K_C} \sigma_{fC}(j)}{K_C} = \\ & \frac{\frac{1}{L_{C1} * \frac{1}{1}} \sum_{j=1}^1 \sigma_{fC1}(j) + \frac{2}{L_{C2} * \frac{2}{2}} \sum_{j=1}^2 \sigma_{fC}(j) + \frac{3}{L_{C3} * \frac{3}{3}} \sum_{j=1}^3 \sigma_{fC3}(j)}{(L_{C1} + L_{C2} + L_{C3})} = \\ & (L_{C1} + L_{C2} + L_{C3}) * \frac{\sum_{j=1}^{K_C} \sigma_{fC}(j)}{L_{C1} * \frac{1}{1} + L_{C2} * \frac{2}{2} + L_{C3} * \frac{3}{3}} \quad (29) \end{aligned}$$

where  $K_C$  means the number of traces in class  $C$  and  $L_{CM}$  means the number of elements in the database with  $M$  traces.

We are only trying to minimize this value, so we can take out the scale factor.

$$\downarrow \frac{\sum_{j=0}^{K_C} \sigma_{fC}(j)}{K_C} \quad (30)$$

$$L_{C1} * \frac{\sum_{j=0}^1 \sigma_{fC1}(j)}{1} + L_{C2} * \frac{\sum_{j=0}^2 \sigma_{fC}(j)}{2} + L_{C3} * \frac{\sum_{j=0}^3 \sigma_{fC3}(j)}{3}$$

That's the case of Length in Stroke, Relative Stroke Length, Accumulated Angle and Quadratic error.

### 3.6.1.5 Vectors with two values for each stroke

When we have a feature which is defined by a vector with the values for two components( $X$  and  $Y$ ) for each trace, we must apply the previous step with the same two-to-one value conversion also mentioned before.

That's the case of the Center of gravity.

### 3.6.2 Weight defining

Once we have computed all we have mentioned before, we have one value for each feature. As we have said, highest values correspond to worst (noisiest) features. So, we must think in a way to express our weights in an inverse proportion to the values we have computed ( $\sigma_f$ ). All weights must sum 1. I used this ponderation.

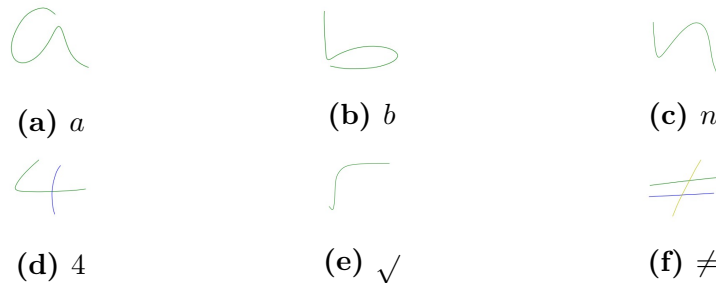
$$w_{fi} = \frac{\frac{1}{\sigma_{fi}}}{\sum_{\forall f} \frac{1}{\sigma_f}} \quad (31)$$

where  $w_{fi}$  means the weight of the feature  $i$ .

### 3.7 Template generation

From the symbols on our database we have to make something that can be compared with the studied symbol. That should be a symbol, then we could compare their similarity.

The most direct way of computing a representing symbol is simply its mean. For every point on the coordinates list in a character, we can compute its mean along the database. We can note that this is the reason of the previous scale normalization.



**Figure 16:** *Some templates*

That leads us to the question: what do we have to do with the rest of features? We have to compute them over the coordinates mean or we have to average them from the symbols of this character on the database? Both options would have sense. I tried them both and I obtained better results averaging again each feature. Indeed this is what I expected, because for this option you don't depend on the coordinates averaging again.

In the special case of the style, which is a quaternary feature (you can only select between four states), I computed it by voting on each character. That means that from each character I stored the most repeated option over its symbols on the database.

#### 3.7.1 Differences on the traces

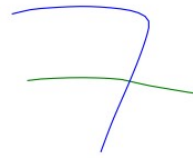
But there's one fact that makes it not so simple. As we have said, this averaging has no sense if we don't compare equivalent points.

Think about a 7. It can be written with only one trace, or with two traces, as we can see in Figure 17. Even that, both cases would be tagged as the same (7). Obviously, the second case has an extra trace that is not equivalent to any of the points on the first one.

We can solve it using the following ideas.



(a) *One trace*



(b) *Two traces*

**Figure 17:** *Different ways to draw a 7, with one or two traces*

### 3.7.1.1 Normalize the number of traces

As the number of traces is the only thing that we have not normalized, we could solve it. As it has been mentioned before (in 3.4.2.8), the size reduction step has been build to join traces in order to reduce the number of them as well as to divide the last trace in order to have more of them. So, we can normalize the number of traces as:

**Its minimum** , which means that if it founds (in the database) a symbol with more traces than the minimum, some of them are joined in order to have less (until this minimum).

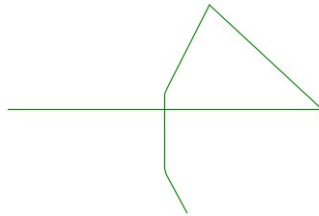
**Its maximum** , which means that for symbols with less traces than the maximum from the character on the database, it adds extra divisions.

But this solution would have remarkable issues. The main one is that the shape will take or forget segments randomly, which would not have any similarity to the original ones. Also, local features like Turning Angle would take wrong values, and global ones like Center of Gravity would be seriously distorted.

### 3.7.1.2 Take those cases as different characters

This method consists on give a different tag for those possible shapes. In fact, we can consider them as different symbols, because when writting it you are choosing one way to describe its shape, and they can be totally different. When building the structure, those tags will mean the same. So, for example, a 7 won't be tagged as 7, but as 71 or 72 depending on its number of traces, and after the classification they will be seen as 7 again. This is the solution I used, for the reasons I have already mentioned.





**Figure 18:** *Effect when two traces are joined, in this case on the symbol +. The resulting shape is wrong*

### 3.7.2 Trace markers normalization

We also have to consider that the length relation for traces on each symbol can be different, and we want to compare equivalent points. That would be an issue if we didn't solve it, because we could be comparing points of different traces.

So, we have to normalize them. That means that traces must be sampled by the same number of points for all symbols on the same character. So, we have to decide again where the traces end for each character.

The simplest solution is to compute the mean between its symbols on the database. For example, imagine a character represented by a symbol with a trace between points 0 and 20 and another one between 21 and 49, and a symbol with a trace between points 0 and 30 and another one between 31 and 49. The resulting trace lengths for the character template would be two traces, one between 0 and 25 and the other one between 26 and 49, which is the mean of the symbols.

When we have decided those limits we can apply an alternative arc length resampling, to adjust the traces on the samples to the decided number of points.

Once we have done all this, we can compute the templates.

### 3.7.3 Independence from shape

There are some special characters on the database. Characters which depend more on their size that on their own shape. That happens when one of the dimensions of the symbol size is very small. Think about the size normalization. When we read a very narrow symbol, we take notice of this fact instead of its local shape, considering it as a vertical line. But our system normalizes it and compares its shape.

Those characters are giving random shapes, or when normalizing its traces order, random orders, so the templates are averaging random traces.

We must specify to the system in which cases that happens, and the way it can solve this issue.

We must delete those characters from the database, and solve this following cases.

### 3.7.3.1 Small horizontal and vertical size

If the system finds a symbol with a small bounding box for both dimensions, it must automatically consider it a dot.

Later, on the structural analysis, depending on where it finds the point it can transform a symbol to another.

### 3.7.3.2 Small vertical size

That's the case of division bars and minus sign  $-$ .

The decision of which of them is the tag is irrelevant, because it will change depending on its structural position.

### 3.7.3.3 Small horizontal size

When the system finds a bounding box with a small width, it must look at its traces.

If it only has one trace, it is considered a 1 (that's the meaning of a vertical line).

If it has two traces, it depends on their length. If the highest one is long (it's considered long if the distance between extremes is more than  $\frac{1}{8}$  of the total bounding box size) it is considered a  $!$ . If the long one is the lowest it is considered an  $i$ . If any of them is long, it's tagged as  $:$ .

Else, it's a strange case. In my system it's tagged as  $:$ .

It's important to remove them from the database not only because their shape is random and it rarely will match them correctly, but also because a lot of false positives can be taken if the given random shape is similar to a concrete symbol.

This is a step of the classification, but it has been explained here to understand why they are removed.

### 3.7.4 Noisy samples

Noisy samples give noisy templates. That's why we must remove those cases, because if not we will be storing wrong templates that will disable the

character itself and can give a lot of false positives.

We can detect noisy samples by its number of traces. In my system, if it detects that from a single character, less than the 6% are written with a concrete number of traces, its considered a rare case, and it's removed. The reason is that if a symbol of 100 samples only has 5 of them written in one trace probably it hasn't been done following regular patterns.

We still have one more problem. There are characters that by definition are written in a concrete number of traces, but has several cases on the database where it's written in more than this number, cut in some point. That point will be random on the shape, so we won't compare equivalent points. I solved it with a manual selection on the database, but that's not the best way to do it.

The rest of noisy samples are also removed manually if they clearly distort the result.

## 3.8 Classification

We have the following: several symbols on a database, tagged by their meaning, which also has a template, and one symbol to classify. We have the features for each of them, and their ponderation.

There are many approaches to this task.

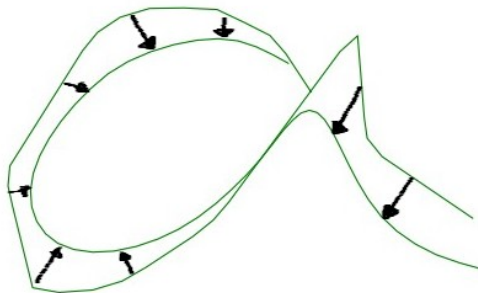
### 3.8.1 Proposed methods

There are proposed in reference [1] some classification methods with different theoretical approaches.

#### 3.8.1.1 Similarity methods

The templates are very relevant here. This is the method that compares them to the studied symbol to simply decide its tag. There is one method called *elastic matching* which is something similar to what I used.

It takes the templates one by one, sums the distances for each point to the studied symbol and stores the result as a cost. Then, he decides the character whose template has less cost. It's normally computed with *DTW* (*Dynamic Time Warping*), which solves the fact that similar point paths can be sampled different, but in our case this is already covered by the arc length resampling.



**Figure 19:** Distance computation between a symbol and a template (*Elastic matching*)

#### 3.8.1.2 Statistical methods

The computed features can be inputs to those classification methods. Neural Networks, HMM (*Hidden Markov Models*) and SVM (*Support Vector Machines*) are proposed.

Neural Networks are proposed to be computed on the structure to compute probabilities on its context, as a kind of feedback to the system. I also tried them directly over the feature vectors, but the result was bad. It's possible that they would be better if I had applied a PCA(*Principal Component Analysis*), which basically is a method that removes correlated information (values on our feature vectors are probably strongly correlated).

HMM are also used on segmentation, so the system would take the information and build the symbols while it's joining the different traces.

SVM would be used only for classification.

### 3.8.1.3 Structural methods

They take each symbol as a combination of shapes. Those shapes are something like *circle* or *-*, and they are predefined on the system. For example, a 6 is formed by a *descending curve* and a *loop*.



**Figure 20:** Example of a character template (6) and the primal shapes that can build it

### 3.8.1.4 Clustering methods

They are proposed to be used as an extra step, to get better results (our classification is supervised, so it wouldn't have sense to use them to classify symbols by tags).

There is an issue to classify a symbol in any conditions, which is the variability of the user handwriting. Clustering methods offer to model those variations and make a better matching.

## 3.8.2 Used method

In my system, I used an algorithm similar to elastic matching, but using not only the coordinates but also the rest of computed features.

Note that before anything, we have to adapt the trace limits of the symbol to the points marked in the template (one adaptation for each template), to compare equivalent points.

My idea was to compute a cost for each feature. As we have said, for coordinates this cost corresponds to the sum of distances for each point between the template and the symbol. For the other features, this cost is also defined as the distance point by point between those two elements. The assigned tag must have the same number of traces, so even in global features it can compare equivalent values (the system assumes that symbols with different number of traces can't have the same meaning and way to be written).

Once we have those costs, how can we use them to make a decision? My first approach was to sum them and look for the lowest cost. But this sum couldn't be done directly.

First of all we should apply the weight of each feature.

Moreover, note that the scale of each feature is very different. For this reason, it is not fair to simply sum all the features, because there would be features more penalized than others. For example, angle related features would be more critical than position related ( $\pi$  against 1), and local features would also be more penalized than global features. One possible solution would be scale them from its maximum possible value:

$$C(s, t) = \sum_{\forall f} w(f) \frac{C_f(s, t)}{\max C_f} \quad (32)$$

where  $C(s, t)$  means the cost of a symbol  $s$  for a template  $t$ , and  $C_f(s, t)$  means the cost of the feature  $f$  of a symbol  $s$  for a template  $t$ .

However, this can be wrong because there are maximums reached easier than other ones (for example, it is very difficult to reach the maximum of coordinates). Then we can normalize again, now by the maximum value of the feature costs found in the database.

$$C(s, t) = \sum_{\forall f} w(f) \frac{C_f(s, t)}{\max(C_f \in dB)} \quad (33)$$

But this comparison would still be not fair, because even normalizing by this found maximum, each feature can have its probability to reach it.

So, we must compute something to make costs penalize a possible decision.

What I did is to maximize the probability, understanding the probability as the sum of the inverse of the costs for each feature divided by the total cost of it (total cost as the sum of this cost for all the templates).

$$Prob(s, t) = \sum_{\forall f} w(f) \frac{1}{\frac{C_f(s, t)}{\sum_s C_f(s, t)}} \quad (34)$$

where  $Prob(s, t)$  means the probability of a symbol  $s$  to be matched with the template  $t$ .

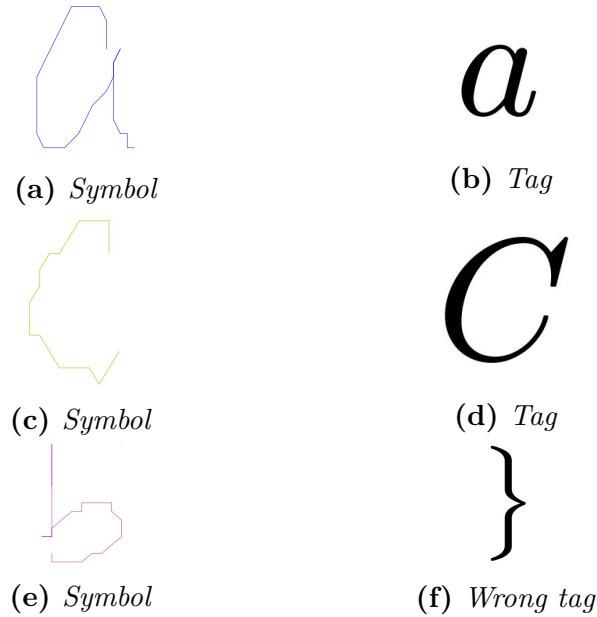
Then, we decide the character of the template with more probability.

Finally we look for special characters as we mentioned before, in section 3.7.3.

### 3.8.3 Results

On the cases I have tested, around a 50% of the symbols are perfectly recognized at this step. This result has improved, because first it was around 33%.

Feature ponderation, cost rescaling redefinition and noisy database elements removal have been the main improvements to achieve this.



**Figure 21:** *Example of some symbols and the character they are tagged on the system*

This result could be improved adding a feedback to the system from the structural analysis, but it has been not implemented yet due to the lack of time of the project.

The main issues are:

- Small but decisive variations between two concrete characters, for example, between ( and [. For us, once we decide that one symbol is one of them, we look at a concrete angle, which is a single component or a

few components of a single feature. Instead of that, the system compares it with all its features in the same way that the rest of symbols, and if the symbol shape is more similar to the wrong template, the decision will fail.

- The way a symbol can be written, not with big differences which can be found on symbols already removed from the database, but variations like the length of a concrete trace which is enough to cause a bad normalization of the symbol. That means that we could be comparing not equivalent points.



## 3.9 Expression building

From this point forward, the system can forget almost everything about the symbols, it only needs to know their found tag and their real ubication (bounding box and center).

Moreover, we don't need all the information in our tag, but only its character, so we can remove its taces number indicator (remeber that for example, 51 means a 5 written in one trace).

So, we have this (a list of tags and the space they fill). What we want is to build a hierarchical structure to describe the expression. For this, is necessary to describe some theoretical concepts to be used on the expression building.

### 3.9.1 Concepts

#### 3.9.1.1 Kinds of symbol

Not all symbols are governed by the same rules. When we humans read, automatically associate some distribution for each symbol, to decide where the following symbols must be to be associated to it or not. And this decision depends on the symbol we are studying. For example, think about a  $c$  and a  $b$ . Note that in  $c^2$  the exponent must be above the character, and if it wasn't we probably would assume it as not associated to  $c$ , while in  $b^2$  the margin is wider, note that the exponent is often located under the  $b$  superior limit. This is because in  $b$  the bounding box fixes the superior limit over where the symbol really concentrates. There are more cases like this, and we can also find the opposite case (for example  $g$ ).

That's why we must define differences depending on the symbol distribution. Basically we can find three cases: central symbols (like an  $o$ ), ascendent symbols (like a  $d$ ) and descendent symbols (like an  $y$ ).

There's one other difference between characters, now more on a semantical sense. Think about a kind of association, like an exponent. Would it mean something associated to a  $+$  ( $+^2$ )? Obviously it wouldn't, so not all the characters must be treated equally. Depending on its possible distribution, we must also classify them.

So, at the end we have a classification like the following, which is what I followed:

	Central	Ascendent	Descendent
Non-scripted	$+, -, /, =, \neq, >, \geq, <, \leq, \dots, !, \exists, \in, \forall, \rightarrow, (, [, \{, \infty$		
Superscripted	$\sin, \cos, \tan, \log, e$	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$	
Scripted	$a, c, e, i, m, n, r, x, z, A, B, C, F, X, Y, \alpha, \beta, \gamma, \theta, \phi, \pm, ), ], \}$	$b, d, f, k, t, \text{f}$	$g, j, p, y$
Sum-like	$\Sigma, \pi$		
Lim-like	$\lim$		
Root-like	$\sqrt{\phantom{x}}$		
Bar-like	$/$		

Table 1: *Classification of characters*

This classification can be decisive in some decisions, because it will define the step that is going to be explained now.

### 3.9.1.2 Symbol regions

To find symbol relations, it's important to define some thresholds that will define the regions for each symbol where other symbols can be located, defining a relation. For example, there is a region for number 2 where another symbol would be its exponent ( $2^x$ ).

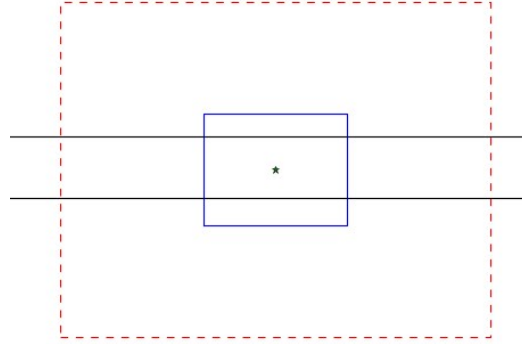
We already have defined one region, which is the bounding box itself. This space is only for the symbol, except for special cases (like the content of a square root  $\sqrt{\phantom{x}}$ ). We also define an extra bounding box, centered on the same point but with triple height and width, called the Outside Box. Symbols directly associated should be inside this last box.

So, associated symbols normally must be between those two boxes, but we also have to define regions inside this. That's why we must define the *superthreshold* and the *subthreshold* values, that will decide how are symbols associated (for example exponents, we will talk about this later). We must also define the *centroid*, that is what will define where a symbol is located to decide if it's associated to some other. Those values will depend on the type of symbol, as it has been already mentioned. I used the following values:

	Central	Ascendent	Descendent
SuperThreshold	$y_s - 0.2H_s$	$y_s - 0.33H_s$	$y_s - 0.1H_s$
SubThreshold	$y_s - 0.75H_s$	$y_s - 0.8H_s$	$y_s - 0.4H_s$
Centroid	$x_s - 0.5W_s, y_s - 0.5H_s$	$x_s - 0.5W_s, y_s - 0.66H_s$	$x_s - 0.5W_s, y_s - 0.33H_s$

Table 2: *Values for thresholds, where  $x_s$  and  $y_s$  are the right and superior limits of the bounding box and  $W_s$  and  $H_s$  the width and height of the bounding box*

So, with this decided we have all the regions for each symbol. Figure 22 shows how are they fit.



**Figure 22:** *Example of region delimiters for a symbol. Blue line is the bounding box, red dashed line is the outside box and black lines are the superThreshold and the subThreshold.*

### 3.9.1.3 Dominances

The dominances is what we use to define those associations between symbols that have been mentioned before.

A dominance is composed by a dominant symbol and a submissive symbol. There are many possible dominances, and we can classify them in two types, hard and soft dominances.

#### Hard dominances

Those which its submissive symbol belongs exclusively to its dominant one. The dominant symbol will include it on his own structure at the end. This is the case of the exponents, for example.

There are many types of hard dominances. Depending on the kind of symbol described before which is the dominant one, the type of hard dominance will be selected from one or another. Those kinds are:

**Superscripts:** exponents, those indicators above on the right from the dominant symbols( $2^x$ )

**Subscripts:** subindices, located below on the right from the dominant symbols( $a_1$ )

**Above:** those indicators simply above from the dominant symbol( $\sum^N$ )

**Below:** those indicators simply below from the dominant symbol( $\lim_{x \rightarrow 0}$ )

**Inside:** those symbols contained inside the dominant symbol( $\sqrt{2}$ )

The possible hard dominances depending on the kind of dominant symbol are the following:

	Superscript	Subscript	Above	Below	Inside
Non-scripted					
Superscripted	✓				
Scripted	✓	✓			
Sum-like			✓	✓	✓
Lim-like	✓			✓	
Root-like	✓		✓		✓
Bar-like			✓	✓	

Table 3: *Possible hard dominances by kind of dominant symbol.*

### Soft dominances

There is only one type of soft dominance, the *right neighbour* relation. By definition, when a symbol is at the same level than the previous one, it is its right neighbour. In the process we must define there are more steps to find them, and they can help finding hard dominances.

Any symbol can be the right neighbour of any other one.

#### 3.9.2 Dominances storing

Our goal is to get all the dominances to build the final structure. We must find a procedure to make a complete analysis with not only the direct dominances. Remember that even out of their regions, there are special rules that could include a dominance over a symbol (for example  $a^{123456}$  includes the 6 on the exponent, which is out of any region), or delete some dominion if a symbol is dominated by two different dominants. I used the following steps. Before anything, we can change some tags depending on the structure. For example if it finds two  $-$  signs aligned in vertical, it can conclude that they are an equal, that would be the case for example if in Figure 23 the equal was segmented as two different symbols. Moreover, if it finds a  $-$  with symbols above and below it can convert it into a division bar. Then we can begin the structural analysis.

First of all, the system looks for direct hard dominances. This is done symbol by symbol, looking to its possible regions. If it finds a symbol whose centroid is on a region to dominate, the system stores a hard dominance where the first symbol is the dominant and the second one is the submissive, of the appropriate type, which is also stored. We can see an example on Figure 23.

$$\sqrt{a^2 + b^2} = c^2$$

**Figure 23:** *First case*

As we can see this case would include superscript hard dominances for  $a$  over first 2, for  $b$  over second 2 and for  $c$  over third 2, and also inside hard dominances of the square root over  $a$ ,  $b$  + and both the first and second 2. After this, the system looks for each symbol the symbol on its right, with its centroid between the limits of its kind (superThreshold or subThreshold, or both, depending on its kind and if it uses them). When it finds that symbol on the right, it looks if it's not dominated by any other symbol which doesn't dominate the first one. If it is not, the system stores a soft dominance where the second symbol is the right neighbour of the first one. If it is already dominated, the system doesn't store any more dominance. For example, on Figure 23, + would be the right neighbour of  $a$ , but the second 2 wouldn't be the right neighbour of the first one, because it's already dominated by  $b$ . It's true that + is also dominated by the square root, but as  $a$  is too, we can conclude that they are at the same level, so they can be neighbours. The soft dominances stored would be from  $a$  to +, from + to  $b$ , from  $\sqrt{\phantom{x}}$  to =, from = to  $c$  and (wrong dominance) from  $b$  to =. There are some cases that have no sense and must be fixed. For example, think on that case:



**Figure 24:** *Duality on possible dominances*

As we can see, directly it would take a subscript dominance of  $a$  over  $\Sigma$  as well as an above dominance of  $\Sigma$  over  $a$ . To avoid this duality, the system must decide one of them. On future improvements, with a probabilistic model it

will decide the most appropriate one, but now I implemented it to keep only the first dominance stored (in that case,  $a_{\Sigma}$ ).

Moreover, it also has to delete those cases where a symbol is right neighbour of two symbols which one is hardly dominating the other one, keeping the neighborhood with the dominant one. This is necessary for something that will be mentioned later. In our example the soft dominance from  $b$  to  $=$  would be eliminated

Now we must define something called *dominant baseline*. The dominant baseline is the set of symbols that are not hardly dominated by any other symbol. The system looks for it at this point. In our example, the dominant baseline would be composed by  $\sqrt$ ,  $=$  and  $c$ .

Then the system looks if some symbol on the dominant baseline is right neighbour of some other which is not on the dominant baseline (that would be the case of our example if we had not deleted the soft dominance from  $b$  to  $=$ ,  $=$  would still be on the dominant baseline because it wouldn't be hardly dominated by any symbol but it would be neighbour of  $b$ , which is not on the dominant baseline). If it finds any case like this, it removes it from the dominant baseline immediately. That's because if it's on the same level than the other symbol, it could not be on the dominant baseline.

Now imagine that our example changes to something like in Figure 25.

$$\sqrt{a^2 + b^2} = c^{2x}$$

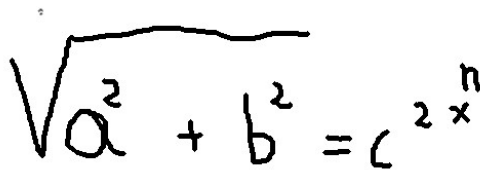
**Figure 25:** *Exponent added*

Note that the new exponent  $x$  wouldn't be taken directly as a superscript of  $c$ , because it does not remain on the superscript region. Even so, we want to add it as an exponent of  $c$ . Note that now the system would have taken one more soft dominance, from 2 to  $x$ . There's the key of our procedure.

We can take every hard dominance, and if the submissive symbol has right neighbours which are not stored as submissive symbols of the same dominance from the dominant one, the system adds it. What means that in our example, as 2 is dominated as a superscript of  $c$  and its neighbour is  $x$ , the system would also store a superscript hard dominance from  $c$  to  $x$ . It repeats this until it doesn't find any more neighbours. If we had not removed the soft dominance between  $b$  and  $=$ , it will assume  $=$  inside the square root.

The next step is to remove useless dominances, as the inside dominances between the square root and both number 2 exponents. The reason is that although for example the first 2 is inside the square root, they relation is not direct, but 2 belongs to  $a$  who belongs to the square root. So it looks symbol by symbol, and if it finds that is hardly dominated by two symbols that also form a hard dominance, deletes its dominance with its grandfather symbol. Then the system removes double hard dominances of one symbol over another one (for example, if on the step of including hard dominances over the neighbours of the submissive symbol of a hard dominance the system had added one hard dominance between two symbols that already had a hard dominance of another type).

Now we only have one more issue. Look at the new example, on Figure 26.



$$\sqrt{a^2 + b^2} = c^{2x^n}$$

**Figure 26:** *Symbol out of ranges.*

The character  $n$  hasn't been taken by any dominance, because it is not on the range of any symbol. To solve this the system adds it to the previous symbol, with the most probable relation. In this case, as it is above the threshold limit of  $x$ , the system stores a hard superscript dominance of  $x$  over  $n$ .

Then, the system repeats the previous steps to avoid paradoxical cases and also the steps of including hard dominances over the right neighbours of submissive symbols and to remove the dominances of grandfather symbols mentioned before.

Finally, the system makes sure than every symbol is hardly dominated by no more than one symbol, and if not, it keeps the first dominance it finds.

The dominances have changed, so it computes again the dominant baseline.

### 3.9.3 Building the expression

At this point we have a list of symbols, a list of dominances and a dominant baseline.

What we must do first is, for each hard dominance, include the submissive symbol as a part of the dominant one. For example, in a  $2^x$ , don't assume

more a 2 itself but a 2 with an exponent  $x$ . And in  $\sqrt{2^x + C}$ , assume it as a square root containing symbols of 2 with an exponent  $x$ , a  $+$  and a  $C$ .

Once the system has assumed that, it can assume the expression as a serie of the symbols which are on the dominant baseline, where each symbol can contain more information.

This expression is easily converted to a  $\text{\LaTeX}$  expression. The result would be something like the following.

$$\sum_{i=0}^n (x_i^{2x} + y_i^{2y})$$

**Figure 27:** *Final result*

### 3.9.4 Results

To test this single step, I faked the classification to be well done so we can focus on the expression building.

Those results were not bad at all. No symbols were repeated or disappeared and normally it returned the general structure of the equation. Some symbols are assigned wrong as superscripts/subscripts/neighbours, and sometimes this means that a fraction structure is broken (for example, if a numerator is taken as a denominator superscript, it can be moved to the lower region of the structure).

There are examples (on Figure 28) of inputs and results.

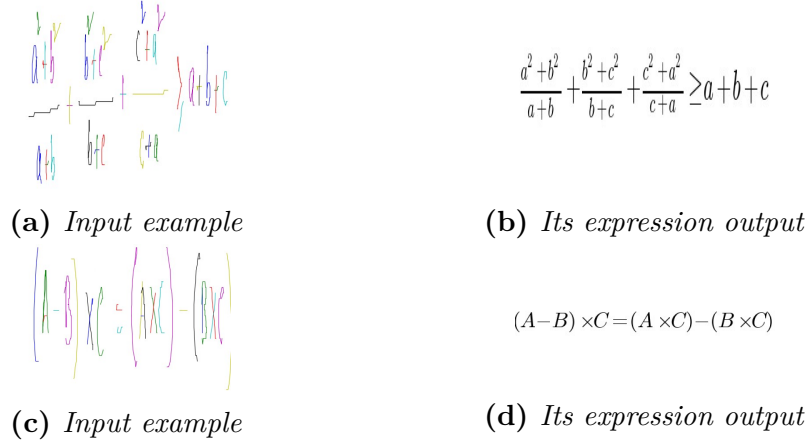
Those results are worse when we don't fake the tags, because the thresholds are distorted and the expression is built wrongly. We will see this results on another section.

### 3.9.5 Improvements on expression bulding: the MST

During my research, I found that it was proposed an implementation of a MST as a probabilistic model.

An MST (*Minimum spanning tree*) is an algorithm that builds a model of relationships that improves the probability of the resulting expression. It defines some nodes, that in this case are the symbols, and the probability of



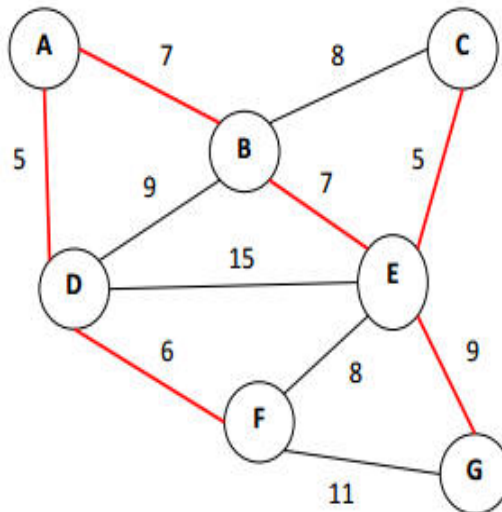


**Figure 28:** Some inputs and their output when their tags are wright

every possible relation. Then, from the most probable relationships builds the complete model.

It is proposed to use the distance between the symbols to compute this probability. Concretely, the distance between the attractor points, which are points defined by the type of symbol and the candidate relationship. It would also compute a dominant baseline and then the sons of its symbols.

That could be a future topic for research, because a probabilistic model could be more robust.



**Figure 29:** Example of a MST


## 4 Results

Many results for each step have been studied on previous sections, but now we must focus on the result for the whole system.

Before studying those results, I want to make clear that this project doesn't present any final version of a program, but a prototype, a first approach to this complex problem. So, the goal is not the accuracy of the results, but the detection of the main issues and the future research topics to improve them. That said, I performed a test weeks before the project end, but I detected some possible sources for the errors and I improved the system. Basically, I implemented some of the steps previously mentioned, such as disabling the shape matching for small symbols (dots, minus signs —...) or removing noisy elements from the database, in order to improve templates sense. I also had to redesign the feature ponderation to the mentioned algorithm.

At the end of the project I performed a test over many cases. I can't report them all on this document, but I will show the most representative ones.

I found many issues of the system while testing it. We can begin with the following case.



(a) *Input*

$$\geq 1 - B_2 + a_{14} - 9^{+z_1 - z_2}$$

(b) *Output*

**Figure 30:** *Wrong classification results on wrong expression*

The most remarkable error detected during the test is the fact that a wrong classification leads to building a wrong expression. In this case on the previous figure, we can see how the decision of tagging the second  $y$  as a 9 moves the thresholds to false values, so their related symbols are assumed with wrong relations. In this case, for example the symbols in the expression  $+z_1 - z_2$  are assumed as exponents instead of as neighbours, as a result of this bad classification. We can see this same effect on the next figure.



(a) *Input*

$$0 + b_{+c + \gamma_{+e}}$$

(b) *Output*

**Figure 31:** *One bad decision is disturbing the rest of the expression*

In this case  $\gamma$  takes its following symbol as a subscript. Note that  $b$  is also taking wrong its relation with its following symbol, although it has been well tagged. That is simply because as you can see,  $b$  is slightly risen from the rest of the expression, so it is considering that its following symbol is low enough for being its subindex. The problem is that, as the rest of symbols are at the same level than  $+$  ( $b$  following symbol), they are assumed as right neighbours of  $+$ , so subindices of  $b$ . This is happening because once the system assumes that  $+$  is the subindex of  $b$ , its relation with the rest of the symbols is based on this previous relation (between  $b$  and  $x$ ). That could be improved setting the direction of the dominant baseline, or instead of directly taking  $c$  as the right neighbour of  $b$ , compute the probability of it against the probability of being the right neighbour of  $b$ .

Now we can focus in another main issue. It is shown on the following figure.



(a) Input

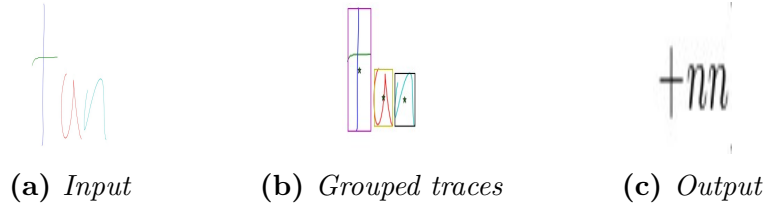
$$(7^4 - B^2 - \beta) \sum zB - 41$$

(b) Output

**Figure 32:** *Intra-class variation is one of the main issues at this point*

This issue refers to the fact that a single character can be written in many ways. Sometimes consciously, sometimes not, this is called Intra-Class variation. The first time I detected the errors it was causing was when I performed my first test. Then was when I made a selection of cases for some characters in the database. On this selection I removed those symbols which were not made with the regular patterns, so the template for those cases had sense again. Even after this selection, the system gave wrong tags, because if the input characters were written in one of those not regular patterns, they were not found on the database. That's the case on the figure, where as we can see those  $x$  are wrongly tagged because their patterns are not those which define  $x$  on its template. This issue could be solved defining one template for each way of writing each character when necessary, as different symbols which mean the same.

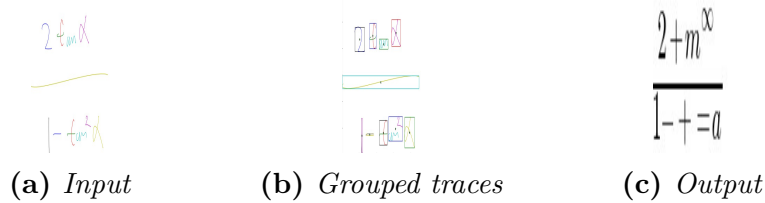
Now we can focus on the last main issue that I found. It is the case of the following figure.



**Figure 33:** *Wrong segmentation can ruin some cases*

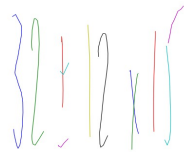
This issue refers to the effect of bad segmentation. Note that on the figure, letters in tan are segmented, so the system is deciding a tag for each isolated symbol. We could implement that the serie of the characters  $t$ ,  $a$  and  $n$  formed the symbol tan when they are found in a concrete position, but as the system is tagging them as other characters, even implementing this we couldn't solve the issue. If we were storing a list of candidate characters instead of making an absolute decision, and also their probability of being joined against being separated, we could get the probability of tan being written against those found isolated symbols. This could be implemented using feedback on the system.

We can see another example of wrong segmentation leading to wrong expression building on the following figure.

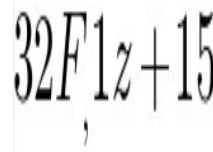


**Figure 34:** *Another case of wrong segmentation effect over classification and expression building*

Finally, we can see an example where a wrong segmentation is making the system tag unwanted symbols that, on the other hand, are wrongly tagged and that gives them a wrong relation. We also can see as subtle variations between symbols such as 2 and  $z$  can lead to bad decisions if features which are irrelevant between them but important in general are more similar to the wrong character. That example could be the summary of this project results.



(a) *Input*



(b) *Output*

**Figure 35:** *Example os the system results*

## 5 Budget

This project has been based on research over software, using *Python*, an Open source programming language with open libraries. The used database is the one offered by *CROHME Competition*, so it is also open.

I have not used any external hardware aside from my personal computers.

The only think to include on this budget is my work time.

### Salaries:

Position	Total number of employees	Hours/month	Amount of months	Total amount of hours	Salary (€/hour)	Total
Junior engineer	1	120	4	480	14	6720
<b>Total</b>	1	120	4	480	14	6720

So we can conclude:

Type of cost	Cost
Salaries	6720 €
Software	0 €
Hardware	0 €
<b>Total</b>	6720 €

The total cost of this project has been **6720 €**.

## 6 Conclusions and future development

The main conclusion is something that has been mentioned previously: This project does not present a final version, and the accuracy of its results is not its goal, in fact we can see that they are not very accurate at this point. Instead, it is presented as a part of a bigger project, which is the Online handwritten mathematical expression recognition. And due to the amount of information it gives we can conclude that it has achieved its main goal. In fact, as its general performance is good in terms of blocks, it can be used as the core of this bigger project.

That said, there are many topics we can develop henceforth. We have mentioned those we have found on our study, but surely there are many more ideas we have not found yet.

There are basically the following kinds of improvements: completing the whole proposed system (adding feedbacks), improving the blocks with more probabilistical models (specially the classification block), defining those ways to write some symbols that have not been included, implementing the support for elements which have not support yet, such as vectors or matrices, and including those blocks not included on the OCR engine, which have not been implemented yet.

It could be also interesting to find some code with the same function and compare it with ours.

### 6.1 Feedback

This should be the following step. The first approach could be assuming the system decision not as a single tag, but as a list of possible characters with its probability. Then, with the structural information, classification could be improved. For example, if a symbol is located on a + sign superscript region, it would consider if the + maybe is another symbol (for example a  $t$ ) instead of assuming this symbol as + right neighbour, depending on its probabilities. This step could be very hard because we should include many thresholds that probably we would obtain by testing the system. This idea could also be used on the segmentation, adapting the structural element of the morphology used to the probabilities of the resulting classification. We can think about the case mentioned before, where a tan was tagged as  $+nn$ , because once their symbols were segmented, the system did not consider that they could be part of a joined symbol.

On my research, I also found some ideas which could be interesting to try. Some authors have used HMM to join traces while the system tries to classify

the symbols. Neural Networks are also mentioned in order to tag symbols using structural information.

## 6.2 Alternative models

I focused in one type of classifier, but there are many kinds that could be tested. Some of them (the statistical models) are mentioned before and include feedback, but structural methods (symbols as composition) and handwriting normalization can also be implemented.

We also can implement the MST building mentioned on the structural step, to make it more probabilistical.

## 6.3 Single character variation

This is one important fact to improve. After the result study, it seems clear that we need to define which writing patterns could a character have, including those which are not the most common.

That could also include an expansion of the database

## 6.4 Supporting more elements

This is probably the less prior step, because before including many options we need a system working. Even that, this could be an interesting topic of research and implementation, and there are many papers related to it.

## 6.5 Outside the OCR engine

As we have mentioned before, this project is focused on the OCR engine. When this engine is ready we will need to include it on the desired application. Depending on the application, other parts can be developed, such as solving the expressions for calculators, formatting and storing for class notes...

In any case we will also need a front-end design.

This is not directly related to this project, but we can use it to get another application.

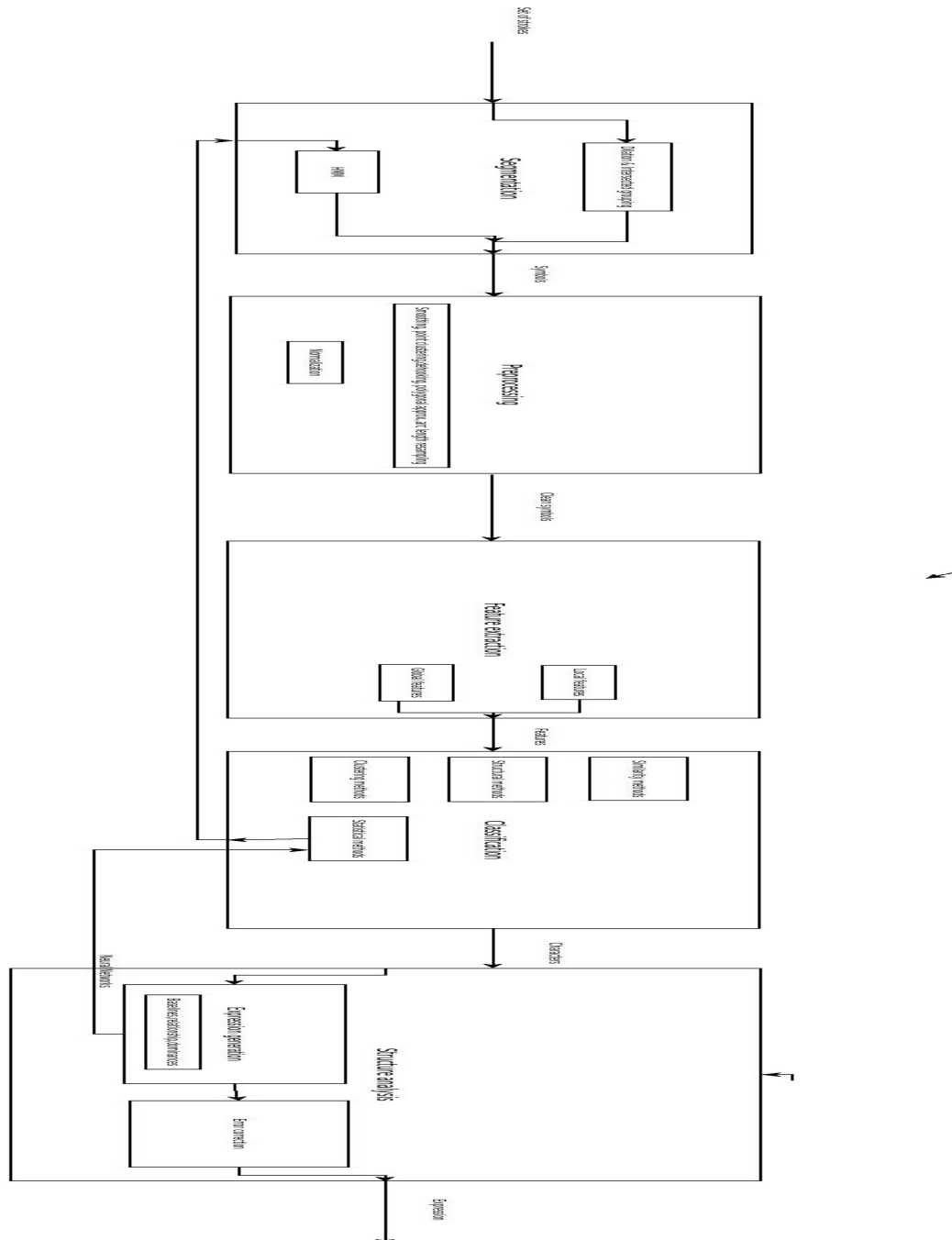


## References

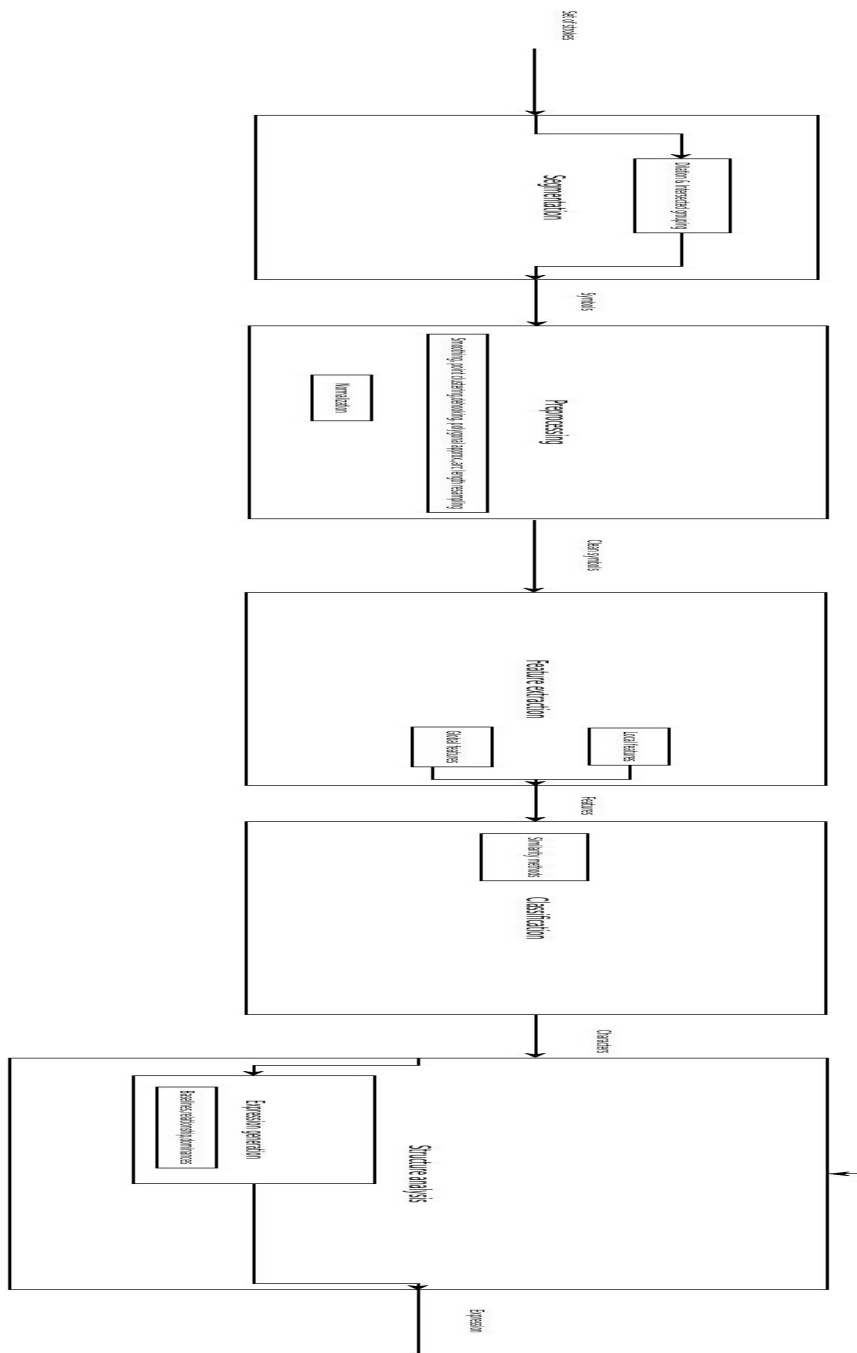
- [1] Ernesto Tapia, Raúl Rojas, *A survey on Recognition of On-Line Handwritten Mathematical notation*, Freie Universität Berlin, 2007
- [2] Kam-Fai Chan, Dit-Yan Yeung, *Recognizing on-line handwritten alphanumeric characters through flexible structural matching*, Department of Computer Science, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, Peoples's Republic of China, 1998
- [3] Chuanjun Li (Brown University), Robert Zeleznik (Brown University), Timothy Miller (Brown University), Joseph J. LaViola Jr. (University of Central Florida) *Online Recognition of Handwritten Mathematical Expressions with Support for Matrices*, 2008
- [4] Kenichi Toyozumi (Nagoya Univ.), Naoya Yamada (Nagoya Univ.), Takayuki Kitasaka (Nagoya Univ.), Kensaku Mori (Nagoya Univ.), Yasuhito Suenaga (Nagoya Univ.), Kenji Mase (Nagoya Univ.), Tomoichi Takahashi (Meijo Univ.), *A study of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information*, 2004
- [5] Francisco Álvaro, Joan-Andreu Sànchez, José-Miguel Benedí, *Recognition of on-line handwritten mathematical expressions using 2D stochastic context-free grammars and hidden Markov models*, Departamento de Sistemas Informáticos y Computación, Universitat Politècnica de València, Valencia, Spain, 2012
- [6] Xiaofang Xie, *On the Recognition of Handwritten Mathematical Symbols*, The University of Western Ontario, London, Ontario, 2007
- [7] Utpal Garain and B. B. Chaudhuri, Fellow, IEEE, *Recognition of On-line Handwritten Mathematical Expressions*, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS, VOL. 34, NO. 6, 2004
- [8] Ernesto Tapia, Prof. Raúl Rojas, Prof. Johan van Horebeek *Understanding Mathematics: A system for the recognition of on-line handwritten mathematical expressions*, Freie Universität Berlin, Berlin, 2004
- [9] *CROHME: Competition on Recognition of Online Handwritten Mathematical Expressions*, [www.isical.ac.in/~crohme/](http://www.isical.ac.in/~crohme/)

- 
- [10] *PyBrain's documentation*, [www.pybrain.org/docs/](http://www.pybrain.org/docs/)

## Appendices



**Figure 36:** *Proposed system*



**Figure 37:** *Implemented system*

Those images are also delivered as *jpg* files on the deliverable appendices.