# Database-Driven Mathematical Character Recognition

Alan Sexton and Volker Sorge

School of Computer Science, University of Birmingham, UK
{A.P.Sexton, V.Sorge}@cs.bham.ac.uk
http://www.cs.bham.ac.uk/~aps|~vxs

**Abstract.** We present an approach for recognising mathematical texts using an extensive LaTeX symbol database and a novel recognition algorithm. The process consists essentially of three steps: Recognising the individual characters in a mathematical text by relating them to glyphs in the database of symbols, analysing the recognised glyphs to determine the closest corresponding LaTeX symbol, and reassembling the text by putting the appropriate LaTeX commands at their corresponding positions of the original text inside a LaTeX picture environment. The recogniser itself is based on a novel variation on the application of geometric moment invariants. The working system is implemented in Java.

## 1 Introduction

Automatic document analysis of mathematical texts is highly desirable to further the electronic distribution of their content. Having more mathematical texts, especially the large back catalogues of mathematical journals, available in rich electronic form could greatly ease the dissemination and retrieval of mathematical knowledge. However, the development of sophisticated tools necessary for that task is currently hindered by the weakness of optical character recognition systems in dealing with the large range of mathematical symbols and the often fine distinctions in font usage in mathematical texts. Research on developing better systems for mathematical document analysis and formula recognition requires high quality mathematical optical character recognition (OCR). As one approach to this problem, we present in this paper a database-driven approach to mathematical OCR by integrating a recogniser with a large database of LaTeX symbols in order to analyse images of mathematical texts and to reassemble them as LaTeX documents.

The recogniser itself is based on a novel application of geometric moments that is particularly sensitive to subtle but often crucial differences in font faces while still providing good general recognition of symbols that are similar to, but not exactly the same as, some element in the database. The moment functions themselves are standard but rather than being applied just to a whole glyph or to tiles in a grid decomposition of a glyph, they are computed in every stage of a recursive binary decomposition of the glyph. All values computed at each level of the decomposition are retained in the feature vector. The result is that the feature vector contains a spectrum of features from global but indistinct at the high levels of the decomposition to local but precise at the lower levels. This provides robustness to distortion because of the contribution of the high level features, but good discrimination power from those of the low levels.

W. Liu and J. Lladós (Eds.): GREC 2005, LNCS 3926, pp. 218–230, 2006.

Since the recogniser matches glyphs by computing metric distances to given templates, a database of symbols is required to provide them. We have developed a large database of symbols, which has been extracted from a specially fabricated document containing approximately 5300 different mathematical and textual characters. This document is originally based on [7] and has been extended to cover all mathematical and textual alphabets and characters currently freely available in LaTeX. It enumerates all the symbols and homogenises their relative positions and sizes with the help of horizontal and vertical calibrators. The single symbols are then extracted by recognising all the glyphs a symbol consists of as well as their relative positions to each other and to the calibrators. Each entry in the database thus consists of a collection of one or more glyphs together with the relative positions and the code for the actual LaTeX symbol they comprise. The basic database of symbols is augmented with the precomputed feature vectors employed by the recogniser.

To test the effectiveness of our OCR system, we analyse the image of a page of mathematics, and reproduce it by locating the closest matching LaTeX symbols and constructing a LaTeX file which can then be formatted to provide a visually close match to the original image. At this stage there is no semantic analysis or syntactic parsing of the results to provide feedback or context information to assist the recognition. As a result, the source produced is merely a LaTeX picture environment that explicitly places, for each recognised character, the appropriate LaTeX command in its correct location. However, it is our position that the information obtained in order to do this successfully is an appropriate input to the higher level analysis required for further document analysis — especially as we can provide, for each glyph, a sequence of alternative characters in diminishing order of quality of visual match. Moreover, the database-driven analysis offers us a way to effectively deal with symbols composed of several, disconnected glyphs, by easily finding, analysing, and selecting all symbols from the database that contain a component glyph that matches the glyph in question.

There has been work on collecting a ground truth set of symbols for mathematics for training and testing purposes. Suzuki et al [13] have compiled a database of symbols, with detailed annotations, from a selected set of mathematical articles. The just under 700,000 characters in their database include many different instances of the same characters, each with true, rather than artificially generated degradation. The actual number of different symbols is much smaller. Our database only contains non-degraded ideal symbols. However, each symbol is generated by a different LaTeX command and so there are relatively few copies of the same glyph in the database. In practice, there is still some duplication in our database because (a) font developers often create new fonts by copying and modifying existing fonts, sometimes leaving some symbols unchanged, and (b) two different multi-glyph symbols often contain copies of one or more of the same component glyphs, e.g. "=" and "≡". Thus Suzuki et al's database is especially suitable for test purposes and for OCR system training on the types of mathematics that appears in the necessarily limited (but large) set of symbols contained in the articles it was extracted from, whereas our database is less suitable for testing purposes but has significantly more breadth in that it contains most supported LaTeX symbols. In particular, we can deal with the rapidly growing number of symbols used in diverse scientific disciplines such as computer science, logics, and chemistry.

A large number of methods for symbol recognition have been studied. See [6] for a high level overview. In our system, which is a development from previous work on font recognition [11], we augment the basic database of symbols with precomputed feature vectors. This database serves as the template set of our character recogniser. The recogniser itself is based on a novel variation on geometric moment invariants. There has been much work on various approaches to character recognition, and geometric moment invariants have been popular [15].

The paper is structured as follows: We present our recogniser and the database of glyphs in Section 2 and 3, respectively. We give an example for the recognition of an involved mathematical expression by our algorithm in Section 4, and conclude in Section 5.

## 2 A Novel Algorithm for Mathematical OCR

Our algorithm for symbol recognition does not depend on a segmentation of the image into full characters. Instead we segment the image into individual connected components, or glyphs, where each symbol may be composed of a number of glyphs. The motivation for this is that because of the 2-dimensional layout in mathematical texts, and in technical diagrams, we do not necessarily have the luxury of having the symbols neatly lined up on baselines. Hence segmentation into symbols is much less reliable in such texts. Instead, our approach is to identify individual glyphs and compose separate glyphs together to form symbols as indicated by the presence of appropriate glyph components in corresponding relative positions.

We assume that preprocessing operations such as deskewing, binarisation etc. have already been applied. The algorithm proceeds by

- extracting glyphs from the image;
- calculating a feature vector for the glyph based on recursive image partitioning and normalised geometric moments;
- for each glyph, producing a list of potentially matching glyphs from the glyph database ordered by metric distance from the target glyph and identifying an initial best match for the glyph.

### 2.1 Extracting Glyphs

Since our feature vector is based on normalised geometric moments of sub-rectangles of the glyph image, we need an appropriate data structure to enable convenient and efficient calculation of such values. Also we need to separate the image into a list of glyph representations, where each glyph is a single connected collection of pixels. We base our moment calculations on that given by Flusser [2], where boundaries of glyphs are used. To take advantage of that, our glyph representation is a list of horizontal line segments, with each segment being represented as the start and end horizontal positions of the row together with the vertical position that the segment occurs at.

Using this representation, the glyphs can be extracted in a single scan down the image. To do so, a set of open glyphs and a set of closed glyphs is maintained. A closed glyph is one which cannot have any further horizontal line segments added to it (because

no such segment could possibly touch any existing line segment in the glyph). An open glyph is one which has at least one line segment on the horizontal row immediately above the one about to be scanned and hence a line segment could be found on the current row which may touch the glyph. In detail the algorithm proceeds as follows:

```
for each horizontal scan line of the image
  for each line segment on the line
    find the set of open glyphs that the line segment touches
    if the set is empty
      create a new open glyph containing the line segment
    else if the set contains only one open glyph
      add the line segment to the open glyph
    else
      merge all open glyphs in the set and add the line segment to the resulting open glyph
  for each open glyph that did not have a line added to it and was not merged
    remove it from the set of open glyphs and add it to the set of closed glyphs
add any remaining open glyphs to the set of closed glyphs; return the set of closed glyphs
```

The above algorithm copes with glyphs with holes e.g. "8", and those which are open at the top, such as "∨" or "⋃". Note that it identifies each of "Θ", "⨦" and "≼" as having two separate glyphs. In particular, this means that, for example, symbols inside frames, such as " ⓐ ", can be handled correctly.

### 2.2 Calculating the Feature Vector

A common approach to statistical symbol recognition is to calculate some properties of a previously segmented symbol, e.g., moments of various orders and types, topological or geometric properties such as numbers of holes, line intersections etc., and to produce a feature vector by collecting the values obtained. The resulting feature vector then can be used, for example, in a metric distance or a decision tree based process to find the best match in a database of symbols. In order to improve classification accuracy using detailed features of the symbol, some systems decompose the image into, typically, a $3 \times 3$ grid of tiles and base the feature vector on calculations on the individual tiles.

Our method also decomposes the image, but instead of into a uniform grid, it decomposes it recursively into sub-rectangles based on the centres of gravity (first order moments) in horizontal and vertical dimensions, such that the number of positive (i.e., black) pixels is equal on either side of the divide. Furthermore, gross features of the image are represented at higher levels of the decomposition while finer details are still captured in the lower levels.

Figure 1 shows an example how a glyph is decomposed. We name the rectangles produced as $R_{i,j}$ where $i \geq 0$ indicates the level of splitting, and $j$ is the component of a split where $0 \leqslant j \leqslant 2^i - 1$. The first two steps and the final result of the splitting are depicted explicitly. $R_{0,0}$ is the entire glyph image. The splitting is binary so $R_{i,j}$ will split into two sub-rectangles $R_{i+1,2j}$ and $R_{i+1,2j+1}$, where these will be the top and bottom (left and right) parts, respectively, if the splitting was using the vertical (horizontal) component of the centroid of the image. The initial splitting is vertical and each level of splitting then alternates between horizontal and vertical. The component of the centre of gravity used to split $R_{i,j}$ we call $y_{i,j}$ if $i$ is even (and the split is therefore

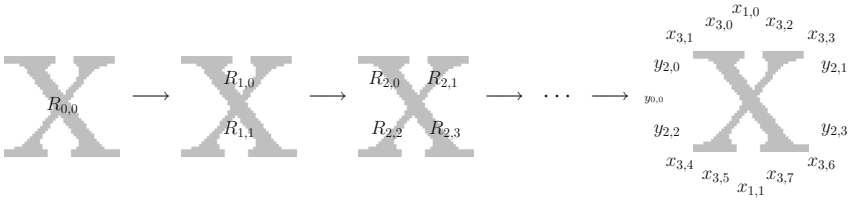


**Fig. 1.** Decomposition split points of "x" at 11 pt

vertical), otherwise $x_{i,j}$ (and the split is horizontal). Note that the regions at any one level are disjoint but are nested within the level above.

For each rectangular region $R_{i,j}$ of a glyph, we calculate 4 feature vector elements: either the vertical or horizontal component of the centroid, $y_{i,j}$ or $x_{i,j}$, (which is later used for further splitting of this rectangle), scaled to a number between 0 and 1, and the three second order scaled central moments, $\eta_{20}, \eta_{11}$ and $\eta_{02}$ [12] for $R_{i,j}$.

In general, the above elements are scale independent to the limits of discretisation errors. We would like to add an extra element based on the aspect ratio. The raw aspect ratio, however, would dominate the vector for tall and narrow or short and wide glyphs, so we use the hyperbolic tan function on the aspect ratio to smoothly map it into a value between 0 and 1. The element is added at the front of the feature vector.

To see this in practice, compare the final decomposition diagram in Figure 1, with the first line of feature vector elements in Table 1 in Sect. 3. The first feature vector element, $fv_0$, is the adjusted aspect ratio just described. In this case, the glyph is 45 pixels wide and 39 pixels tall so $\tanh(39/45) = 0.70$ to 2 decimal places.

The next four elements are derived from $R_{0,0}$, i.e., the rectangle described by the outer bounding box of the entire glyph. The vertical centre of gravity is indicated by the $y_{0,0}$ line. This is very slightly above the geometric centre of the image and so is shown, in Table 1, as 0.49 for $fv_1$ (in line with Java image processing, the origin of the image is the top left corner and the y coordinates increase down the image). $fv_2$, $fv_3$ and $fv_4$ correspond to $\eta_{2,0}, \eta_{1,1}$ and $\eta_{0,2}$ for $R_{0,0}$. The next four elements, $fv_5, \ldots fv_8$, are the corresponding elements for $R_{1,0}$, the top half rectangle of the image. Here $fv_5$ is marked in the figure as $x_{1,0}$, which, because of the greater thickness of the upper left arm of the glyph, is to the left of the middle of that rectangle with a value of $0.46$. The vector elements for the lower half, $R_{1,1}$ follow next. Then comes, in order, the top left, the top right, the bottom left and the bottom right rectangle and so on recursively.

Extraction of the glyph from the document has produced a representation based on lists of line segments, which is suitable for efficient calculation of moments via boundary analysis [8, 2]. Furthermore, calculation of the moments for the sub-rectangles does not require complicated or costly construction of new line segment lists but instead is carried out by simply limiting the values of the line segments to that which would appear in the required rectangle when executing the calculation.

In our current implementation, we are using 4 levels of splitting, resulting in 15 regions, from which we extract feature vector elements and hence the vector currently contains 61 elements. We are experimenting with feature selection techniques to choose a suitable smaller subset of the features for actual metric function evaluation.

### 2.3 Initial and Alternative Matching Glyphs

Given a database of glyphs with associated precomputed feature vectors, we use the standard euclidean metric for computing glyph similarity. We collect a list of the best matches which can then be filtered to remove obvious impossibilities, apply feedback constraints from higher levels of the document analysis process and impose priorities on choosing between a collection of database glyphs that are all of similar metric distance from the target glyph. The best glyph resulting is returned as the best match but the list is retained so that higher levels of the system could potentially reject the first choice and recalculate the best match based on extra feedback information, e.g. contextual information returned from a semantic analysis.

Obvious impossibilities occur when a possible matching glyph is one component of a multi-glyph symbol but the other components of the same symbol are not found at the corresponding relative location in the target document. Feedback constraints from higher level processing could include, for example, that a particular character appears to be a letter in a word which, by dint of dictionary lookup, is likely to be one of a very small number of letters in a particular font. Priorities that can be applied include giving higher priority to recognising a symbol at a particular font size, over one at a different font size but scaled to match the target glyph.

At this level our approach is purely syntactic. This leaves us with the semantic problem of recognising symbols consisting of disconnected glyphs. While we could leave this to a later stage, in practice we believe this unnecessarily complicates the task of higher level processing. We instead can take advantage of the database information to notice when we have a good match on a component of a symbol and directly search for the remaining components.

We currently use a double track approach: (a) If the glyph matches with a symbol that consists of that one glyph alone we can simply pick it (the result may not be the correct symbol from a semantic point of view but the formatted output should be visually indistinguishable). (b) In the case that the best match for a recognised glyph is a glyph in the database that belongs to a symbol that is composed of multiple glyphs we cannot simply take that symbol since it might introduce glyphs into the result that have no counterpart in the original document. In this case we can consider two possible conflict resolution strategies:

1. We search all closely matching glyphs for one that is the only glyph of its associated symbol.
2. We search all closely matching glyphs for one whose sibling glyphs in its symbol are also matched in the appropriate relative position.

While approach 1 might not necessarily deliver the best matching glyph, it definitely will not introduce superfluous information into the document. But in some cases it will not be possible to find a symbol that matches acceptably well with the original glyph and approach 2 might be preferable (and in general, approach 2 is, of course, more correct from a semantic perspective), which forces a search over sets of glyphs of the particular area under consideration. In our current (first) implementation we give preference to approach 1 by allowing for a small error threshold when matching glyphs and giving a preference to matching single glyph symbols over multi-glyph symbols within that threshold. If this fails, however, we do resort to approach 2.

For example, consider the symbols "⨦" and "≼" from Section 2. The former can be composed by two separate symbols, one for "⋂" and one for the inlaid "+". For the latter, however, there is no appropriate match such that we can compose the "≼" symbol from two separate, single glyph symbols. While we can find a match for "≺" that is the only glyph of its symbol, the only matches available for the curly upper bar in "≼" belong to multi-glyph symbols. Therefore, the algorithm searches for a symbol whose glyphs match as many other glyphs as possible surrounding the curly upper bar in the input image. In the case of our example the algorithm would indeed come up with "≼" as the closest matching symbol.

## 3 The Symbol Database

The templates for the system are held in a database of approximately 5,300 symbols, each in 8 different point sizes, that is augmented with precomputed feature vectors for the character recognition process. We shall briefly describe the design of the database (for a more detailed description on how the database is constructed, see [9]) and explain its main characteristics with an example.

In detail, the database currently consists of a set of *LaTeX formatted documents* (one per point size for 8, 9, 10, 11, 12, 14, 17 and 20 points), which are rendered to tiff format (multi-page, 1 bit/sample, CCITT group 4 compression) at 600dpi. The documents are originally based on [7] and have been extended to cover all mathematical and textual alphabets and characters currently freely available in LaTeX. However, the database can be easily extended for more symbols by adding them to the documents. The documents enumerate all the symbols and homogenise their relative positions and sizes with the help of horizontal and vertical calibrators. The single symbols are then extracted by recognising all the glyphs a symbol consists of as well as their relative position to each other and to the calibrators. We store them in a suitable directory structure with one tiff file per glyph, and no more than 100 symbols per directory, together with an index file containing the requisite extra information such as bounding box to base point offsets, identification of sibling glyphs in a symbol, precomputed feature vector, etc.

In addition to these documents we have an *annotation text file* that is automatically generated from the LaTeX sources during formatting and that contains one line for each symbol described in the LaTeX documents, which associates the identifier of the symbol with the LaTeX code necessary to generate the symbol together with the information on what extra LaTeX packages or fonts, if any, are required to process the code and whether the symbol is available in *math* or *text* mode. Thus we can view each entry in the database as a collection of one or more glyphs together with the indexing information and the code for the actual LaTeX symbol they comprise.

The character recognition extracts single glyphs from the document under analysis and then tries to retrieve the best matches from the database. Since all this works on the level of glyphs only, we take a closer look at the information on single glyphs that is stored in the index files. This information consists essentially of three parts: (1) basic information on the overall symbol, (2) basic information on the glyph, and (3) the precomputed feature vector containing all possible moments described in Section 2.

Information of type (1) and (2) is mainly concerned with bookkeeping and contains elements such as width, height, absolute number of pixels of a symbol or glyph, re-

spectively as well as indexing information. The feature vector (3) is then the actual characterising information for each glyph. It is pre-computed with a uniform length for all glyphs in the database at database creation time. In the current version of the database the vector contains 61 different features. But despite the length of the vector not all features must necessarily be used by the recogniser. In fact the algorithm can be parameterised such that it will select certain features of the vector and restrict the matching algorithm to compute the metric only with respect to the features selected. This facilitates experimenting with combinations of different features and fine-tuning of the character recognition algorithm without expensive rebuilds of the database.

**Table 1.** Feature vectors for the symbols "x", "$\mathsf{x}$", and "$\times$"

| | $fv_0$ | $fv_1$ | $fv_2$ | $fv_3$ | $fv_4$ | $fv_5$ | $fv_6$ | $fv_7$ | $fv_8$ | $fv_9$ | $fv_{10}$ | $fv_{11}$ | $fv_{12}$ | $fv_{13}$ | $fv_{14}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0.7 | 0.49 | 0.18 | 0.02 | 0.3 | 0.46 | 0.34 | 0 | 0.1 | 0.5 | 0.37 | -0.01 | 0.11 | 0.33 | 0.15 |
| $\mathsf{x}$ | 0.75 | 0.49 | 0.18 | 0 | 0.27 | 0.48 | 0.35 | 0 | 0.1 | 0.48 | 0.39 | 0 | 0.12 | 0.44 | 0.16 |
| $\times$ | 0.76 | 0.49 | 0.32 | 0 | 0.32 | 0.5 | 0.63 | 0.01 | 0.14 | 0.5 | 0.63 | -0.01 | 0.14 | 0.47 | 0.29 |

| | $fv_{15}$ | $fv_{16}$ | $fv_{17}$ | $fv_{18}$ | $fv_{19}$ | $fv_{20}$ | $fv_{21}$ | $fv_{22}$ | $fv_{23}$ | $fv_{24}$ | $fv_{25}$ | $fv_{26}$ | $fv_{27}$ | $fv_{28}$ | $fv_{29}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0.1 | 0.16 | 0.34 | 0.22 | -0.19 | 0.26 | 0.62 | 0.23 | -0.2 | 0.27 | 0.6 | 0.15 | 0.12 | 0.18 | 0.49 |
| $\mathsf{x}$ | 0.16 | 0.21 | 0.44 | 0.16 | -0.16 | 0.21 | 0.52 | 0.17 | -0.19 | 0.24 | 0.52 | 0.19 | 0.2 | 0.24 | 0.36 |
| $\times$ | 0.28 | 0.29 | 0.49 | 0.29 | -0.28 | 0.29 | 0.48 | 0.29 | -0.28 | 0.29 | 0.46 | 0.29 | 0.28 | 0.29 | 0.25 |

| | $fv_{30}$ | $fv_{31}$ | $fv_{32}$ | $fv_{33}$ | $fv_{34}$ | $fv_{35}$ | $fv_{36}$ | $fv_{37}$ | $fv_{38}$ | $fv_{39}$ | $fv_{40}$ | $fv_{41}$ | $fv_{42}$ | $fv_{43}$ | $fv_{44}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0.3 | 0.02 | 0.03 | 0.76 | 0.1 | 0.08 | 0.15 | 0.53 | 0.21 | -0.02 | 0.04 | 0.19 | 0.18 | -0.17 | 0.22 |
| $\mathsf{x}$ | 0.14 | 0.07 | 0.09 | 0.73 | 0.11 | 0.09 | 0.12 | 0.57 | 0.13 | -0.07 | 0.09 | 0.23 | 0.11 | -0.09 | 0.12 |
| $\times$ | 0.17 | 0.12 | 0.13 | 0.7 | 0.18 | 0.15 | 0.16 | 0.72 | 0.16 | -0.13 | 0.14 | 0.26 | 0.18 | -0.14 | 0.15 |

| | $fv_{45}$ | $fv_{46}$ | $fv_{47}$ | $fv_{48}$ | $fv_{49}$ | $fv_{50}$ | $fv_{51}$ | $fv_{52}$ | $fv_{53}$ | $fv_{54}$ | $fv_{55}$ | $fv_{56}$ | $fv_{57}$ | $fv_{58}$ | $fv_{59}$ | $fv_{60}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x | 0.75 | 0.18 | -0.18 | 0.24 | 0.39 | 0.21 | -0.03 | 0.05 | 0.19 | 0.1 | 0.08 | 0.13 | 0.5 | 0.22 | 0.03 | 0.05 |
| $\mathsf{x}$ | 0.75 | 0.11 | -0.09 | 0.14 | 0.36 | 0.13 | -0.09 | 0.11 | 0.22 | 0.12 | 0.1 | 0.13 | 0.6 | 0.14 | 0.09 | 0.11 |
| $\times$ | 0.72 | 0.17 | -0.13 | 0.15 | 0.27 | 0.18 | -0.14 | 0.14 | 0.24 | 0.17 | 0.13 | 0.14 | 0.7 | 0.18 | 0.14 | 0.16 |

As an example of feature vectors we compare the symbols "x", "$\mathsf{x}$", and "$\times$", given in the annotation text file as the LaTeX commands `\textrm{\char'170}`, `\textsf{\char'170}`, and `$\times$`, respectively. Each of the symbols only consists of one glyph, whose feature vectors are given in Table 1. Note that the single component values are given with two digit precision only, while in the actual database the numbers are given with full double float precision.

If we now, for instance, consider the first fifteen features, we can observe that there is very little difference between the values for "x" and "$\mathsf{x}$". However, both differ quite considerably from "$\times$" in features $fv_2, fv_6, fv_{10}, fv_{14}$ (i.e., in the $\eta_{2,0}$ moments for the whole glyph, the top and bottom halves of the glyph and the top left quarter of the glyph). This indicates that "$\times$" has essentially the same symmetries as the other two symbols but that the pixels in the corresponding sub-rectangles are more spread out horizontally around the respective centres of gravity for the "$\times$" symbol than for the other two. We can find the first distinguishing features for the two symbols "x" and "$\mathsf{x}$" in the vector component $fv_{13}$. This is a first order moment corresponding to the centre of gravity $y_{2,0}$ in Figure 1. It basically reflects the impact of the top left hand serif in "x", which pushes the centre of gravity upwards and therefore results in a smaller value than for the other two symbols.

If we now want to compare the three symbols with each other using the entire feature vector, we can compute the following Euclidean distances for the three pairs of symbols:

$$|\mathrm{x} - \mathsf{x}| = \sqrt{\sum_{i=0}^{60} (fv_i(\mathrm{x}) - fv_i(\mathsf{x}))^2} \approx 0.47066 \qquad |\mathrm{x} - \times| \approx 0.83997 \qquad |\mathsf{x} - \times| \approx 0.62672$$

The numbers indicate that "$\mathsf{x}$" is a closer match for "x" than "$\times$". However, the "$\times$" is naturally still closer to the sans serif "$\mathsf{x}$" than to the "x" with serifs. However, none of the three symbols is actually a close match for any of the others, since while the distances between symbols can theoretically range between $0$ and $\sqrt{61} \approx 7.81$, overall a distance $\leqslant .15$ is generally considered a close match by the recogniser when using this set of features.

## 4 An Example Recognition

In order to evaluate the effectiveness of our approach we have essentially two methods to assemble documents: Firstly, we take the closest matching glyph image from the database, possibly apply some scaling to it, and place it at the position in the new document that corresponds to the position of the recognised glyph in the original file. While this has the advantage that we can directly use single glyphs recognised and retrieved from the database and therefore do not have to deal with symbols consisting of several disconnected glyphs, it has the disadvantage that the resulting file is in a picture file format that cannot be used for further processing.

The second method is to assemble an actual LaTeX source file that formats to the recognised text. For this, the glyphs in the original document are identified and an appropriate symbol is chosen from the database. The LaTeX command for that symbol is then put at the correct position in the output document within a LaTeX picture environment whose measurements correspond to the bounding box given by the original document. The restrictions imposed by LaTeX on the `unitlength` of the picture environment can affect the exact placement of characters, since commands can only be put at integer raster points in the environment. Scaling is applied by specifying and selecting a particular font size for the LaTeX command, which also imposes some restrictions with respect to the available font sizes.

We demonstrate the results of our algorithm with an example from a paper, [4], we have experimented with that offers a large number of complex mathematical expressions. The particular expression we are interested in is given in Figure 4 as it appears in the paper. Note that it is in its original form in 10 point size font. As comparison, the results of the OCR algorithm are displayed in Figures 2 and 6, where the former contains the assembled tiff file and the latter the formatted LaTeX generated expression. Since the results are difficult to distinguish with the naked eye, we have combined images 4 and 2 using *exclusive-or rendering* in Figure 3. The similar combination of images 4 and 6 is given in Figure 5. Here, all pixels that show up in only one of the two images appear as black pixels.

The difference in the rendering is more severe for the generated LaTeX expression than for the assembled tiff file. This is due to the fact mentioned earlier, that the characters cannot be placed exactly at the right positions but only approximately at the next

$$\begin{aligned}[\![A \to B]\!] &= S \to [\![A]\!] \to (S \times [\![B]\!]) \\ &\cong \prod_{w' \in \mathcal{W}} (Sw' \to [\![A]\!] \to \sum_{w'' \in \mathcal{W}} (Sw'' \times [\![B]\!]))\end{aligned}$$

**Fig. 2.** Generated mathematical expression image

**Fig. 3.** Difference between Figure 4 and Figure 2 using XOR rendering

$$\begin{aligned}[\![A \to B]\!] &= S \to [\![A]\!] \to (S \times [\![B]\!]) \\ &\cong \prod_{w' \in \mathcal{W}} (Sw' \to [\![A]\!] \to \sum_{w'' \in \mathcal{W}} (Sw'' \times [\![B]\!]))\end{aligned}$$

**Fig. 4.** Original mathematical expression image

**Fig. 5.** Difference between Figure 4 and Figure 6 using XOR rendering

$$\begin{aligned}[\![A \to B]\!] &= S \to [\![A]\!] \to (S \times [\![B]\!]) \\ &\cong \prod_{w' \in \mathcal{W}} (Sw' \to [\![A]\!] \to \sum_{w'' \in \mathcal{W}} (Sw'' \times [\![B]\!]))\end{aligned}$$

**Fig. 6.** LaTeX generated mathematical expression

possible integer raster point. Since the algorithm computes the character positions from the centre outwards, in the LaTeX expression the symbols in the middle have the most overlap and the discrepancy increases towards the outside.

But also the generated tiff image does not match the original expression exactly. There are essentially three types of differences which are best explained when looking at the LaTeX code of the original expressions:

```
\newcommand{\seman}[1]{ [\![ {#1} ]\!] }
\begin{eqnarray*}
  \seman{A  \rightarrow B} & = &
  S \rightarrow \seman{A} \rightarrow (S \times \seman{B}) \\
  & \cong &
  \prod_{w'\in {\mathcal{W}}}(Sw' \rightarrow \seman{A} \rightarrow
  \sum_{w''\in {\mathcal{W}}}(Sw'' \times \seman{B}))
\end{eqnarray*}
```

Firstly, we observe that the author of [4] did not use pre-designed symbols for the semantic brackets but rather defined them via a new, handcrafted macro as an overlap of the regular square brackets. Nevertheless, the recogniser finds a suitable replacement for the characters, for instance, the `\textlbrackdbl` command in the **textcomp** package for the left semantic brackets. The distance between the original expression

and the symbol in the database is $0.0689$. This discrepancy is essentially caused by the slightly leaner vertical bars in the `\textlbrackdbl` command, which can indeed be observed in the respective feature vectors. While nearly all features differ by a minimal fraction, only, the features $fv_{33}$ and $fv_{45}$, corresponding to the vertical splits $x_{3,1}$ and $x_{3,4}$, respectively, have a difference of roughly $0.03$.

Secondly, the difference in the "$\in$" is caused by the recogniser retrieving the command `\ABXin` from the **mathabx** package in 17 point size, which, however, does not scale as well to the required font size of 9 points as the original "$\in$" symbol would.

Finally, the remaining differences are essentially due to scaling to a point size for which no corresponding characters in the database exist. Our current database does not contain a 7 point set of symbols. However, the "$w$" and the calligraphic "$\mathcal{W}$" in the subscript of the sum and product symbol are actually in 7 point size. The closest match in the database yields the 9 point versions. These have to be scaled to size 7, which leads to slight discrepancies in the rendering. In the generated LaTeX expression this scaling is achieved, for instance, by LaTeX command `{\fontsize{7}{0}\selectfont $\mathnormal{w}$}`.

## 5 Conclusion

We have presented a novel algorithm for mathematical OCR that is based on a combination of recursive glyph decomposition and the calculation of geometric moment invariants in each step of the decomposition. The current implementation of the algorithm yields nearly optimal results in recognising documents that are already compiled from actual LaTeX source files. We are currently experimenting with scanned images of documents, in particular, we have started experimenting with articles from the Transactions of the American Mathematical Society [14]. Within the repository of the JSTOR archive [3], images of all the back issues of this journal — starting 1900 — have been made available electronically. While the results of these experiments are already encouraging, more experimentation with feature selection and fine tuning of the recognition algorithm is needed to achieve a robust top quality recognition.

We compared the results from our recogniser with those from two other recognisers we call *Box* and *Grid*. Both use the same aspect ratio feature and the same moment functions as our recogniser, but Box includes the moment functions only on the entire glyph and Grid includes them for each of the 9 tiles of a $3 \times 3$ grid subdivision of the glyph. Preliminary results indicate that the Box recogniser performs worst, presumably due to the lack of sensitivity to details of the glyphs. The Grid recogniser suffers from the arbitrary nature of features of empty or near-empty cells in the grid (e.g., the upper right cell of an 'L' character) — a disadvantage that our system is not subject to.

The effective limit on the recursive decomposition of the glyphs to extract feature vectors is the increasing discretisation errors that arise as we try to decompose rectangles with fewer and fewer positive pixels. In practice, for any rectangle, the calculation of the geometric moments will gather some discretisation error, as discussed in [5]. This error can be reduced, at some computation cost, by being more precise in how one translates from the proper continuous integral expression for the moment calculation to the discrete version for a binary image. However, another form of discretisation error

appears as we split into sub-rectangles based on rounding the split point to the nearest pixel boundary. We are currently investigating the costs and benefits of applying a more accurate approach to the discretisation problem here.

The success of our approach depends on the availability of a large high quality database of symbols generated from a large set of freely available LaTeX fonts. In its current version, the database contains, among its approximately 5,300 symbols, 1,600 mathematical symbols and 1,500 characters from different mathematical alphabets. The remaining symbols are mostly regular textual characters, accents, as well as additional scientific symbols, such as chemical or meteorological symbols. Since we keep copies of each symbol at 8 different point sizes, we are currently storing about 42,400 different symbols in total. Since many symbols are composed of more than one glyph, and we actually store glyphs rather than symbols in the database (but with sufficient information to reconstruct the full symbols as needed), we are actually storing about 59,000 glyphs. Nevertheless, the database is easily extensible and is therefore also suitable for recognising scientific texts other than mathematics.

Analysing a document involves extracting the glyphs from the document and finding its nearest neighbours with respect to the metric in the database. The nearest neighbour search is searching in the full database of the 59,000 glyphs for each target glyph in the system. On a moderately powerful desktop PC running Linux, the software, in its current unoptimised state takes about 10 minutes to process a page of three to four thousand target glyphs. Many optimisations are possible to improve this speed but, to provide true scalability to very large databases of symbols, we intend to use an SM-tree [10], a high performance variant of the M-tree metric file access method [1]. However, for our current work, we are using a naïve internal memory algorithm which is slower but adequate for non-production use and easier to experiment with.

## References


1. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the 23rd VLDB Conference*, p.426–435, 1997.
2. J. Flusser. Fast calculation of geometric moments of binary images. In M. Gengler, editor, *Pattern Recognition and Medical Computer Vision*, p.265–274. ÖCG, 1998.
3. The JSTOR scholarly journal archive. `http://www.jstor.org/`.
4. P. Levy. Possible world semantics for general storage in call-by-value. In J. Bradfield, editor, *Proc. of CSL'02*, volume 2471 of *LNCS*, p.232–246. Springer, 2002.
5. W. Lin and S. Wang. A note on the calculation of moments. *Pattern Recognition Letters*, 15(11):1065–1070, 1994.
6. J. Lladós, E. Valveny, G. Sánchez, and E. Martí. Symbol recognition: Current advances and perspectives. *LNCS* 2390, p.104–127, 2002.
7. S. Parkin. The comprehensive latex symbol list. Technical report, CTAN, 2003. available at `http://www.ctan.org`.
8. W. Philips. A new fast algorithm for moment computation. *Pattern Recognition*, 26(11):1619–1621, 1993.
9. A. Sexton and V. Sorge. A database of glyphs for ocr of mathematical documents. In Michael Kohlhase, editor, *Proc. of MKM-05*, LNCS. Springer Verlag, 2005. In print.
10. A. Sexton and R. Swinbank. Bulk loading the M-tree to enhance query performance. In *Proc. of BNCOD-21*, volume 3112 of *LNCS*, pages 190–202. Springer Verlag, 2004.

11. A. Sexton, A. Todman, and K. Woodward. Font recognition using shape-based quad-tree and kd-tree decomposition. *Proc. of JCIS-2000*, p.212–215. Assoc. for Intel. Machinery, 2000.
12. M. Sonka, V. Hlavac, and R. Boyle. *Image processing, analysis and machine vision*. International Thomson Publishing, 2nd edition, 1998.
13. M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. In *Proc. of ICDAR-2005*, p.675–679. IEEE Computer Society Press, 2005.
14. Transactions of the American Mathematical Society. Available as part of JSTOR at `http://uk.jstor.org/journals/00029947.html`.
15. D. Trier, A. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey,. *Pattern Recognition*, 29(4):641–662, 1996.