Vectorial Signatures for Symbol Discrimination

Philippe Dosch, Josep Lladós

Vectorial Signatures for Symbol Discrimination

Philippe Dosch* and Josep Lladós+

* LORIA, *615, rue du jardin botanique, B.P. 101, 54602 Villers-lès-Nancy Cedex, France*
E-mail:Philippe.Dosch@loria.fr

+ *Centre de Visió per Computador, Edifici O, Campus UAB, 08193 Bellaterra (Cerdanyola), Barcelona, Spain*
E-mail:josep@cvc.uab.es

Abstract

In this paper, we present a method based on vectorial signatures, which aims at discriminating, by a fast technique, symbols represented within technical documents. The use of signatures on this kind of document has an obvious interest. Indeed, considering a raw vectorial description of the graphical layer of a technical document (e.g. a set of arcs and segments), signatures can be used to perform a pre-processing step before a "traditional" graphics recognition processing, or can be used to establish a classification that can be sufficient to feed a further indexation step. To compute vectorial signatures, we have based our approach on a method proposed by Etemadi et al., who study spatial relations between primitives to solve a vision problem. We consider five types of relations, invariant to transformations like rotation or scaling, between neighboring segments: parallelism with or without overlapping, collinearity, L junctions and V junctions. A quality factor is computed for each of the relations, computable with low requirements of power. The signature of all models of symbols that could be found in a given document are computed and matched against the signature of the document, in order to determine what symbols the document is likely to contain. The quality factor associated with each relation is used to prune relations whose quality factor is too low. We present finally the first tests obtained with this method, and we discuss the improvements we plan to do.

Keywords: symbol discrimination, symbol recognition, vectorial signatures, technical drawings

1 Introduction and related works

In this paper, we present a method based on vectorial signatures, which aims at discriminating symbols represented in technical documents. Within the field of pattern recognition, signatures belong to the large family of pattern numerical descriptors. These descriptors are often used on bitmap images to efficiently analyze some structural features, allowing a statistical classification.

Among the global descriptors, some descriptors like moments invariants [5] or Zernike moments are useful to describe the global shape of a pre-segmented pattern. Some comparative studies of these moments can be found in [9] and [1]. To the opposite of these approaches, a geometrical invariant description can be computed by using some primitives, which are locally very informative. These local descriptors appear to be easy to use and relatively robust, in particular to handle occlusions, junctions, etc. [8] Close to this problem, some members of our teams work in particular on the definition of discrete signatures [7, 12].

In graphics recognition, high level processes for the analysis and recognition of the document components are usually formulated in terms of vectorial representations. Thus, signatures in terms of raw images as described above are not suitable, but vectorial signatures are required. However, the literature seems to be a bit poor in the field of vectorial signatures, although the use of signatures on this kind of document has an obvious interest. Indeed, systems based on signature techniques are generally fast, as they rely on relatively basic computations. Considering a raw vectorial description of the graphical layer of a technical document (e.g. a set of arcs and segments), signatures can be used to perform a pre-processing step before a "traditional" graphics recognition processing, this kind of methods being very time and CPU consuming in some cases. They can also be used to establish a classification that can be sufficient to feed a further indexation step.

In our knowledge, few papers present the use of vectorial signatures for recognition or discrimination needs. Approaches for iconic indexing in image databases in terms of a 2D string encoding can inspire the statement of vectorial signatures. It is the case in particular for the system of Huang [4] which improves the indexation of some large databases with this kind of techniques. These databases contain several thousands of pictures, composed of several basic distinct objects. One the one hand, each picture is indexed according to an object-oriented image knowledge structure called 2D C^+-string, which is computed with respect to 13 defined spatial relations between the objects of the images. The retrieval of a picture with this structure is then computed by a strings subsequence matching. However, this matching is time consuming as the retrieval involves the check of the string representing the query with each of the strings representing the pictures. So, on the other hand, some of the basic objects are chosen to be the keys used to build some signatures. The choice of these keys depends of the frequency of the objects occurrence in the database pictures. When a query occurs, the signature of the query is matched with the image signatures, which are then mapped to the corresponding 2D C^+-strings. This allows to decrease the complexity of the retrieval with respect to the straight retrieval based on 2D C^+-strings.

Ventura and Schettini [11] use vectorial signatures to directly recognize symbols. They work on technical documents created with CAD software. These documents are binarized, segmented (text/graphic and thick/thin lines), skeletonized and vectorized. From the thin segments resulting from the vectorization, they extract some features as the number of segments intersecting in one point, the acute angles between segments, the length... From the thick segments, they extract the area, the orientation and second order moments. All these features are combined to create symbols signatures. Signatures are then organized into a structured filter used for the recognition task. Two values are associated with each features of the signature: a tolerance threshold and a weight. A candidate symbol is then analyzed in the same way. Its signature is compared with the signatures of the symbols library. For each feature of the signatures, its value must be lower than the associated tolerance threshold. Then, each parameter is normalized using the weight, and the sum is compared to a global threshold. If the sum is lower than this threshold, the candidate matches the symbol.

Thus, even if few methods deal with vectorial signatures, it appears that they can be employed both for optimization and recognition purposes. The remainder of this paper is organized as follows. Section 2 presents the objectives of our method, the framework and the assumptions made. Section 3 describes how the vectorial signatures are computed from the vectorial description of an image. The implementation side of this method, giving some optimization tips, is studied in section 4. In section 5, we present how these signatures are used to discriminate symbols. Finally, results and perspectives are discussed in section 6.

2 Objectives and framework

By considering vectorial signatures, our intention is not to deal with the hard problem of symbol recognition. It is essentially to discriminate symbols by a fast technique, rather than to recognize them, as this latter task can be approached by some well-known techniques in document analysis. The advantages of using vectorial signatures in the context of symbol recognition are twofold. First, it allows to reduce the computation time of the symbol recognition stage that usually is a time-consuming process. Second, it solves the symbol segmentation problem, i.e. sometimes the type of documents does not allow to segment symbols to their recognition because they appear embedded in the document. The vectorial signature allows to define regions of interest likely to be a given prototype symbol.

Few assumptions have been made, as our method intends to be as generic as possible. We consider the family of technical documents, which are essentially composed of graphical information. We especially work on a raw vectorial description of these documents, that is supposed to be a set of arcs and segments. It is also assumed that these documents usually include several representations of a predefined number of symbols, whose construction follows some structural rules. However, no assumption is made about possible transformations (rotation, scaling) between the symbols models and their instances in a document. Note that, as this paper describes a work in progress, the segments alone are considered in this paper.

3 Toward vectorial signatures

3.1 Introduction

A signature of an entity may be defined as a set of elementary features, containing intrinsically a discrimination potential. In some cases, this allows several entities to share the same signature, even if a unique signature for each entity is desirable. In the particular context of graphics recognition, a symbol signature is computed from the primitives composing the symbol (some pixels if one works on a bitmap image, or some vectorial primitives if the input data is a vectorial image). In our case, we work on the vectorial description of technical documents.

To compute vectorial signatures, we have based our approach on the method proposed by Etemadi *et al.* [3], who study spatial relations between primitives to solve a vision problem.

The method is interesting as:

- The methods start by a study of basic relationship between pairs of lines. Several main relations are thus enumerated: collinearity, parallelism and intersections. For each of these relations, some extensions are considered, like overlapping for parallelism, or the kind of intersection point (lying on either, both or only one segment).
- Their approach leads to a consistent formation of a signature. According to the relations definition, a pair segment can belong to only one of the definition. It is extremely useful that the relations are defined according to some thresholds (on length angles, and so on...)
- By grouping these basic relations, they compute some higher level relations, expressing more symbolic relations between sets of segments. Thus, the approach is clearly hierarchical.
- The relations are associated with a quality factor, computed as a comparison between a given pair of segments and their grouping in an ideal case.
- Finally, the proposed method is invariant to affine transformations, like rotation, scaling or translation.
- Even if the method is designed for a computer vision purpose, a few assumptions are made on the segments. Thus, the method seems to be suitable for our needs.

3.2 Basic features computation

Our current work is based on this method. Of the seven proposed relations of the article, we consider only five (see the "Junction detection" section). These five types of relations express relationship between neighboring segments, and are computable with low requirements: parallelism with or without overlapping, collinearity, L junctions and V junctions, described below. These relations are successively studied in a way presented by the figure 1.

Some of them require some thresholds, which are presented below:

- σ^{θ} is the threshold used to determine whether two segments are considered parallel (as well as not parallel). Thus, if the acute angle θ between two segments satisfies: $|\theta| < \sigma^{\theta}$, the two segments are considered as parallel. Note that, as mentioned by Etemadi *et al* [3], there is no restriction for the choice of the threshold. Indeed, whatever the value, the parallelism relation, as it is defined, is consistent. This allows to tune this value, according to the quality of input graphical primitives, as well as the expected quality of the signatures.
- $\sigma_i^{\parallel}$, as introduced by Etemadi *et al*, is called the *standard deviation of the position of the end points of a segment S_i along its direction*. It allows to take into account the location uncertainties associated with the segment extraction process. In our case, this process is the vectorization, which delivers segments provided with their thickness (let T_i be this thickness for the segment S_i). As the vectorization step

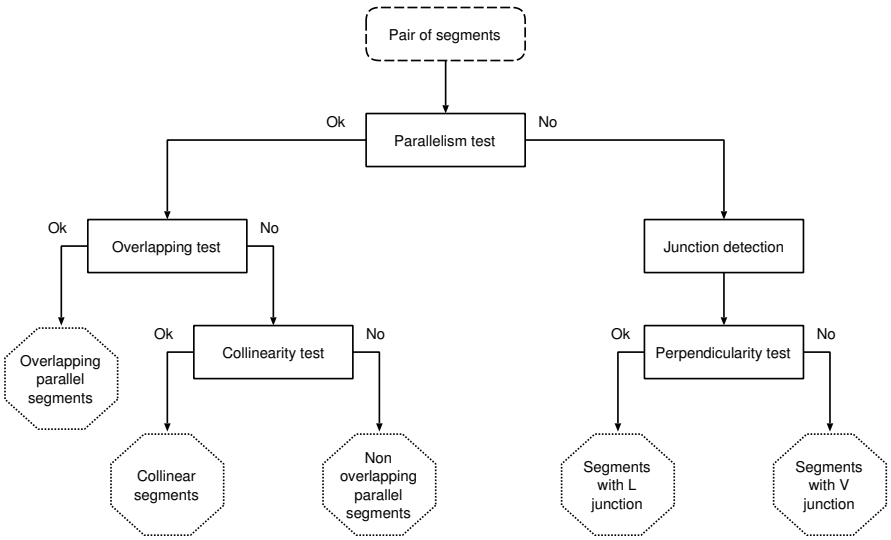

Figure 1: Determination of the relationship between a pair of segments.

may cause junction misplacements, proportional to the thickness of the strokes, $\sigma_i^{\parallel}$ is defined as: $\sigma_i^{\parallel} = \left\lceil \frac{T_i}{2} + 1 \right\rceil$, where $\lceil \rceil$ expresses the rounded up notation.

- $\sigma_i^{\perp}$, as introduced by Etemadi *et al*, is called the *standard deviation of the position of the end points of a segment S_i perpendicular to its direction*. It corresponds to the maximum perpendicular distance allowed between S_i and another segment collinear to S_i. In this case, the tolerance has to be greater than in the precedent case, and $\sigma_i^{\perp}$ is defined as: $\sigma_i^{\perp} = \left\lceil \frac{T_i}{2} + 3 \right\rceil$.

Let us further describe the different types of relationships between pairs of segments. But before presenting them, we want to point out some useful remarks about the segments. In our analysis, segments result from a vectorization step [10] which delivers sometimes spurious short segments, especially whenever thick lines are present in the original document. These artifacts, as all short segments, are not considered in our method, as their consistency is very low. So we consider only segments larger than a given threshold (experimentally fixed to 10 pixels in our case) as input data of figure 1.

3.2.1 Parallelism test

As said previously, if the acute angle between two segments is below the chosen threshold σ^{θ}, the segments are considered as *parallel*. Note that if this property is fulfilled, it does not ensure that a relation will be created between the segments. Indeed, according to the figure 1, some other tests have to be performed to determine what is the exact type of the relation, which will be finally created if the segments satisfy some quality requirements.

3.2.2 Overlapping test

To perform this test between two segment S_1 and S_2, we first determine the virtual line VL lying between the two considered segments. Let L_i and θ_i be respectively the length and the orientation angle of S_i. The orientation angle θ_{VL} of the virtual line is computed as the weighted mean of the orientation of the two segments, as:

$$\theta_{VL} = \frac{L_1 \times \theta_1 + L_2 \times \theta_2}{L_1 + L_2}.$$

Let $M_i(x_i, y_i)$ be the midpoint of the segment S_i. The point $P(x_{VL}, y_{VL})$ through which the virtual line passes is defined by (see figure 2):

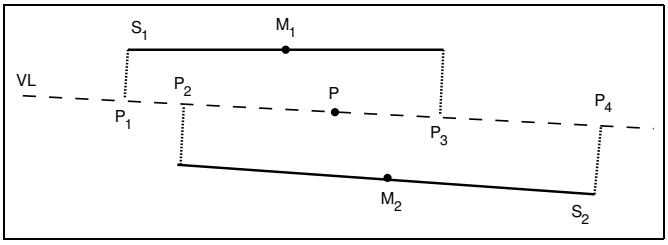

Figure 2: Representation of the overlapping relation.

$$x_{VL} = \frac{L_1 \times x_1 + L_2 \times x_2}{L_1 + L_2} \text{ and } y_{VL} = \frac{L_1 \times y_1 + L_2 \times y_2}{L_1 + L_2}.$$

Once the virtual line is computed, the points P_1, P_2, P_3 and P_4 can be computed by a perpendicular projection of the endpoints of S_1 and S_2 on the virtual line. From the virtual line, a virtual segment VS is defined as the longest segment that it is possible to define with the points P_1, P_2, P_3 and P_4 (which is $[P_1, P_4]$ on figure 2). Let L_{VS} be the length of the virtual segment VS, and L_{P_i} the projected length of the segment S_i on the virtual segment. Now, the two segments S_1 and S_2 are considered as *overlapping* segments if:

$$L_{VS} \leq L_{P_1} + L_{P_2} + \sigma_1^{\parallel} + \sigma_2^{\parallel}.$$

The quality factor Q_{OVP} of the overlapping relation is obtained by computing:

$$Q_{OVP} = \frac{L_{P_1} + L_{P_2} - \sigma_1^{\parallel} - \sigma_2^{\parallel}}{2 \times L_{VS}}.$$

Thus, a perfect quality factor of 1 is obtained if the sum of the two projected length is equal to the double of the length of the virtual segment.

3.2.3 Collinearity test

If a pair of segments are parallel without overlapping, these segments can be classified as non-overlapping parallel segments or collinear segments. According to the definition of $\sigma_i^{\perp}$, if the perpendicular distance between two segments S_1 and S_2 is greater than $\sigma_1^{\perp} + \sigma_2^{\perp}$, the segments are classified as non-overlapping parallel segments. Otherwise, they are classified as *collinear* segments. In both cases, the quality factor Q_{NOVP} is defined as:

$$Q_{NOVP} = \frac{L_{P_1} + L_{P_2} - \sigma_1^{\parallel} - \sigma_2^{\parallel}}{L_{VS}}.$$

Thus, a perfect quality factor of 1 for either a non-overlapping parallel or a collinear relation is obtained if the sum of the two projected length is equal to the length of the virtual segments.

3.2.4 Junction detection

From a mathematical point of view, a pair of non-parallel segments is a pair of segments that intersect. In our case, we consider that a pair of segments intersect if the acute angle between them is equal or greater to σ^{θ}. This threshold is the same as the one used to test the parallelism, in order to ensure the consistency between the different relationships.

As noticed previously, short segments are not considered in our analysis. But it appears that junctions are often represented after the vectorization step by a short segment joining 2 other segments. By choosing to not

consider short segments, the junction between two segments may be lost, as well as the associated symbolic information. So, our approach is to make no difference between segments that really intersect (that share an endpoint) and segments that *virtually* intersect, even if we test the existence of a possible shared point in order to avoid useless computations. If no shared point is found, we compute the intersection point of the segments by extending the segments as shown on figure 3.

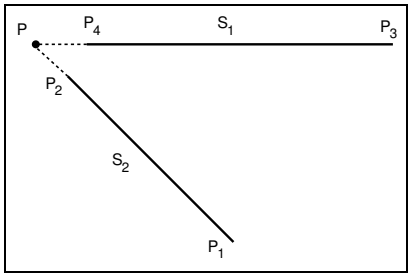

Figure 3: Representation of the junction relation.

We call P the intersection point between the two (virtually extended) segments. The quality factor Q_J is computed as follows:

$$Q_J = \frac{L_1 - \sigma_1^{\parallel} - \sigma_2^{\perp}}{L_{VL1}} \times \frac{L_2 - \sigma_2^{\parallel} - \sigma_1^{\perp}}{L_{VL2}}.$$

Thus, the more the segments are virtually extended, the more Q_V decreases. If only one segment (say L_1) has to be extended in order to find the intersection point, the quality factor Q_J is computed as:

$$Q_J = \frac{L_1 - \sigma_1^{\parallel} - \sigma_2^{\perp}}{L_{VL1}}.$$

This special case of intersection could produce some new types of relations. In particular, Etemadi *et al* define two relations called λ and T junctions. But it appears that these relations do not represent relevant information when dealing with technical documents. Indeed, these relations are usually detected on special cases, corresponding to vectorization artifacts. Thus, for our needs, no discrimination potential is expressed by using them with technical documents. This is probably related to the processing chain which is different between computer vision and technical document analysis, and in particular to the way the chaining process deals with junction points. And anyway, even for Etemadi *et al*, few relations of these two types are found on their examples.

3.2.5 Perpendicular test

Once the junction is computed, we classify it according to the acute angle between the two segments. If this angle is close to $\frac{\pi}{2}$, more or less σ^{θ}, we consider that the junction is perpendicular (*L junction*). Otherwise, the junction is classified as non-perpendicular (*V junction*).

3.2.6 Quality factor

For each relation among the 5 types defined above (parallelism with or without overlapping, collinearity, L junctions and V junctions), a quality factor has been defined to express the pertinence of the relation (e.g. perfect overlapping for parallelism with overlapping). In order to keep only a set of relevant relations for the next steps, non-relevant relations, that is to say the relations whose the quality factor is lower than a fixed threshold Q_F, have to be pruned. The value of Q_F has to be fixed with care: choosing a too high value will result in few produced relations and choosing a too low value will result in too many produced relations. In both cases, the set of produced relations is not suitable enough to be used as a signature.

3.3 Clustering

Even if these basic relationships represent interesting discrimination features, more interesting ones can be computed by clustering the relations. Indeed, a cluster of relations of the same type is more symbolic and relevant than the same set of individual relations. So, once all the basic relations are computed, and are pruned using the quality factor, a simple clustering is performed. The *signature* of an entity is then defined as the set of clustered features and unclustered relations found in that entity. For the purpose of symbol discrimination, two kinds of entities are handled: the references symbols, cropped directly from an image (see section 5 for details) and the buckets of a test image (see section 4 for more explanations about buckets). Briefly, these buckets can be defined as small rectangular parts of the test image.

Considering all the relations of an entity, the relations of the same type are incrementally clustered. Starting from a relation of a type $\mathcal{T}$, containing two segments S_1 and S_2, we search for the relations where either S_1 or S_2 is a part of another relation of the same type $\mathcal{T}$. All the relations satisfying this constraint are then merged into a new cluster. The process is iterated, considering the new set of relations of the current cluster, until no more relation can be added to this cluster. No new quality factor is computed for the clusters, as the relations of the clusters have already been filtered using a quality factor, and as the matching step essentially lies in an inclusion test (see section 5).

4 Implementation side

In our case, implementation is very important as the method has to be very time-efficient in order to be as interesting as possible. Remember that one of the main application is to constitute a pre-processing step before a "traditional" recognition step. In this case, the time spent with the signature system must be less important than the same recognition system without signatures. Another possible case of use is the interactive recognition of symbols, after an on-line selection of a symbol of interest. Here again, the time-efficiently is very important, as interactive applications have to be reactive enough.

So, if the method and the algorithms are naturally important, so is the implementation. We have chosen to implement the signature system with C++ and the STL, known to be very efficient development tools. One of the most time expensive step in the method described above is the computation of neighboring relations. Without any particular data structure, this computation for n segments leads to $\frac{n^2-3n-2}{2}$ computations. To reduce the complexity, we use a bucket structure, which allows just to locally study the relationship between pairs of segments. This kind of technique is used for example in [2].

This technique, very basic, consists in a decomposition of the image in several non overlapping tiles. The graphical primitives are then stored in these buckets, and relationships are only computed between the primitives of a bucket and the primitives of its neighboring buckets. As the objective of signature is to discriminate symbols, the size of the buckets can be based on the size of the biggest symbol available. Let m be the number of buckets. Depending on the repartition of the graphical primitives on the image, the complexity can be reduced to $\frac{n^2-mn-m^2}{2}$ computations (considering an homogeneous distribution). As a typical example, a bitmap image of 1685×1583 pixels is converted into 450 segments and is decomposed with 256 buckets, decreasing the number of computations by a factor of 10.

We plan to use a more sophisticated bucket structure, as the ones described in [6] for example, in order to decrease the complexity more significantly.

5 Discrimination using symbol signatures

These signatures are used in the following way. Using the method described above, we compute the signature of all symbols we can find in a given document. The resulting signatures are stored in a symbol library. The representation of each symbol is actually directly cropped from an original document – that is to say we do not use ideal models for symbol representation (see figure 4 for examples). The underlying idea is to allow a user to

(a) (b) (c) (d) (e) (f)

Figure 4: Examples of reference symbols, cropped directly from an architectural drawing. (a) A door. (b) A window. (c) A French window. (d) A table. (e) A bed. (f) Another kind of table.

select an area from a document including a symbol, from which all symbols having the same representation are found. Also, the vectorized representations of the reference symbols have to be built with the same processes as the test images, in order to reproduce the same artifacts, and then similar signatures. The number and the type of the relations found for each symbols of figure 4 are presented in table 1.

Symbol	Overlap. Paral.	Non overlap. paral.	Collinear	L junction	V junction
(a)	1	0	0	0	4
(b)	6	0	0	0	0
(c)	3	1	0	8	10
(d)	2	0	0	4	0
(e)	6	4	2	10	15
(f)	2	3	0	4	12

Table 1: Number of relations found per type from the symbols of the figure 4, using a quality factor equal to 0.5. Note that the symbols have been vectorized before the computation of the relations.

Then, considering the vectorial representation of a test image, the corresponding graphical primitives (segments) are stored in buckets, whose size is set to half the size of the biggest symbol available. The signature of each bucket is computed, determined by the set of relations found in it. As for the symbols signatures, a quality factor is used to prune the relations whose relevance is too low. The signatures of the buckets are then matched against the symbols signatures to determine what kind of symbols a bucket is likely to contain.

The matching is implemented as an inclusion test. For the signature of a given bucket, the signature of each reference symbol is considered. The signature of a reference symbol $\mathcal{S}$ is included in the signature of the bucket $\mathcal{B}$ if:

- Each basic (individual) relation of $\mathcal{S}$ matches a distinct basic relation of $\mathcal{B}$ of the same type.
- Each clustered relation $\mathcal{R}_\mathcal{S}$ of $\mathcal{S}$ matches a distinct clustered relation $\mathcal{R}_\mathcal{B}$ of $\mathcal{B}$ of the same type, containing *at least* as much relations as in $\mathcal{R}_\mathcal{S}$.

In these rules, "distinct" means that a given relation of the bucket can be matched with at most one relation of the studied symbol. If this inclusion test succeeds, the signature of the bucket contains the signature of the symbol, and thus the bucket is likely to contain the symbol.

The first tests are encouraging, even if we consider only a minimal set of features to compute the signatures. The figure 5 presents results obtained from a vectorial representation of an architectural drawing, with a symbol library containing symbols (a), (b) and (c) of the figure 4. In this figure, the areas of interest (buckets), determining the potential presence of architectural symbols (doors, windows...), are presented by a box containing the name of the symbol(s) detected. Some comments about these results are:

- All symbols are detected with the good type, except a window at the bottom right.

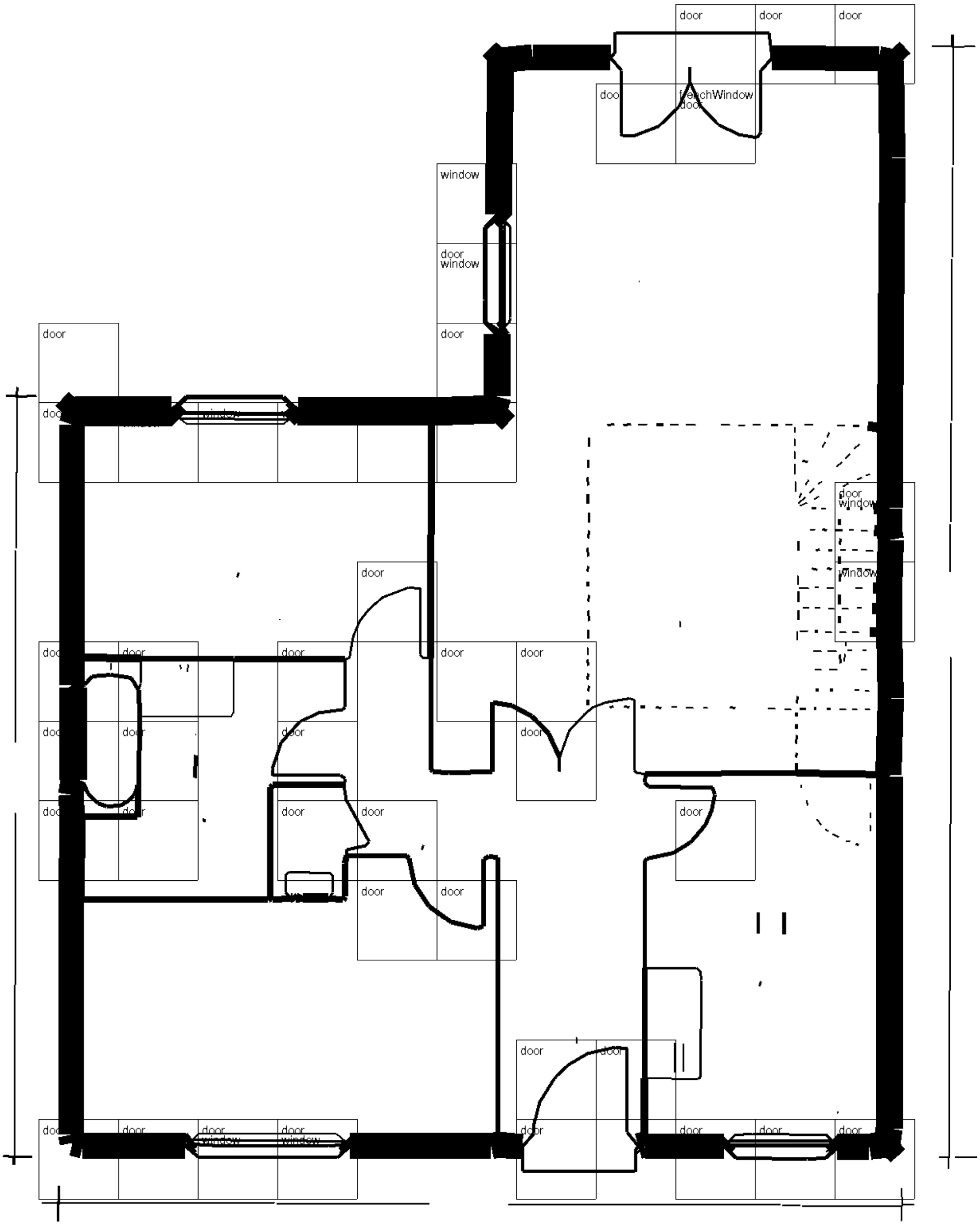

Figure 5: Some results obtained by vectorial signatures computation on an architectural drawing. The areas of interest, determining the potential presence of architectural symbols (doors, windows...), are presented by a box containing the name of the symbol(s) detected.

- A lot of false alarms are present, especially with symbols not present in the library (like the bathtub) and with corners that, after the vectorization, have a representation close to the representation of the doors (see figure 6).

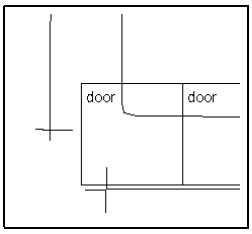

Figure 6: Skeleton representation of the bottom left corner of the house of the figure 5 after vectorization.

- Symbols containing some arcs (typically, doors) often lead to non relevant signatures, depending on the way the arcs are approximated during the vectorization step.
- Since reference symbols have small dimensions, the length of some segments resulting from the vectorization is lower than 10 pixels. It distorts the resulting set of relations.

Thus, it appears obvious that arcs have to be taken into account to avoid some of these false alarms and that we have to work with a vectorization which delivers segments with more accurate locations.

6 Conclusion and further work

As said previously, this paper describes a work in progress, and a lot of improvements can be done. We plan to improve the management of neighboring relations, in particular the definition and the use of buckets, as the speed is a critical aspect of the process. We also work on a better definition of the relations, including other primitives (arcs), width of primitives, arity of primitives, more sophisticated possibilities of clustering... Few symbols have been considered during our tests. If the number of symbols grows, it may also be interesting to "organize" the symbols signatures, e.g. in a hierarchical way, in order to optimize the matching step.

We wish to point out that most of the work needed to compute the relations could be done in a previous process (e.g. vectorization step), making this method even faster. Finally, as claimed in the introduction of this paper, we have to measure the potential use of this method to improve a technical document recognition chain.

All implementations have been realized using the Qgar software package, available at `http://www.qgar.org`.

References

[1] S. O. Belkasim, M. Shridar, and M. Ahmadi. Pattern Recognition with Moment Invariants: A Comparative Study and New Results. *Pattern Recognition*, 24:1117–1138, 1991.

[2] R. C. Bolles and R. A. Cain. Recognising and Locating Partially Visible Objects: the Local-Feature-Focus Method. In A. Pugh, editor, *Robot Vision*, pages 43–82. Springer-Verlag, Berlin, 1983.

[3] A. Etemadi, J-P. Schmidt, G. Matas, J. Illingworth, and J. Kittler. Low-level Grouping of Straight Line Segments. In *Proceedings of Second British Machine Vision Conference, Glasgow, Scotland*, pages 118–126, 1991.

[4] P.W. Huang. Indexing Picture by Key Objects for Large-Scale Image Databases. *Pattern Recognition*, 30(7):1229–1237, 1997.

[5] I. Rothe, H. Susse, and K. Voss. The Method of Normalization to Determine Invariants. *IEEE Transactions on PAMI*, 18(4):366–379, 1996.

[6] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[7] S. Tabbone and L. Wendling. Technical Symbols Recognition Using the Two-dimensional Radon Transform. In *Proceedings of the 16th International Conference on Pattern Recognition, Québec (Canada)*, volume 2, pages 200–203, August 2002.

[8] T. Taxt, J. B. Olafsdottir, and M. Daehlen. Recognition of Handwritten Symbols. *Pattern Recognition*, 23:1155–1166, 1990.

[9] C. Teh and R. T. Chin. On Image Analysis by the Method of Moments. *IEEE Transactions on PAMI*, 10(4):496–513, 1988.

[10] K. Tombre, C. Ah-Soon, Ph. Dosch, G. Masini, and S. Tabbone. Stable and Robust Vectorization: How to Make the Right Choices. In A. K. Chhabra and D. Dori, editors, *Graphics Recognition—Recent Advances*, volume 1941 of *Lecture Notes in Computer Science*, pages 3–18. Springer-Verlag, September 2000.

[11] A. Della Ventura and R. Schettini. Graphic Symbol Recognition using a Signature Technique. In *Proceedings of the 12th International Conference on Pattern Recognition, Jerusalem (Israel)*, volume 2, pages 533–535, 1994.

[12] L. Wendling, S. Tabbone, and P. Matsakis. Fast and robust recognition of orbit and sinus drawings using histograms of forces. *Pattern Recognition Letters*, 23(14):1687–1693, 2002.