International Journal on Document Analysis and Recognition (IJDAR) manuscript No.
(will be inserted by the editor)

Generation of Synthetic Documents for Performance Evaluation of Symbol Recognition & Spotting Systems

Mathieu Delalandre[1], Ernest Valveny[1], Tony Pridmore[2], Dimosthenis Karatzas[1]

[1] CVC, Barcelona, Spain
[2] SCSIT, Nottingham, England

xx-xx-2008/ xx-xx-2009

Abstract This paper deals with the topic of performance evaluation of symbol recognition & spotting systems. We propose here a new approach to the generation of synthetic graphics documents containing non-isolated symbols in a real context. This approach is based on the definition of a set of constraints that permit us to place the symbols on a predefined background according to the properties of a particular domain (architecture, electronics, engineering, etc.). In this way, we can obtain a large amount of images resembling real documents by simply defining the set of constraints and providing a few predefined backgrounds. As documents are synthetically generated, the groundtruth (the location and the label of every symbol) becomes automatically available. We have applied this approach to the generation of a large database of architectural drawings and electronic diagrams, which shows the flexibility of the system. Performance evaluation experiments of a symbol localization system show that our approach permit to generate documents with different features, that are reflected in variation of localization results.

1 Introduction

This paper deals with the topic of performance evaluation. Performance evaluation is a particular cross-disciplinary research field in a variety of domains such as Information Retrieval [1], Computer Vision [2], CBIR[1] [3], DIA[2] [4], etc. Its purpose is the development of frameworks to evaluate and compare a set of methods in order to select the best-suited for a given application. Due to the heterogeneity of the fields where performance evaluation can be applied to [1–4], it is difficult to find a common definition of "what a performance evaluation framework is". Two main tasks are usually identified (Figure 1): groundtruthing, which provides the test and reference data to be used during the evaluation (both for training and testing), and performance characterization, which determines the metrics and the protocol to match the results of a given method with the groundtruth in order to give a measure of the performance.

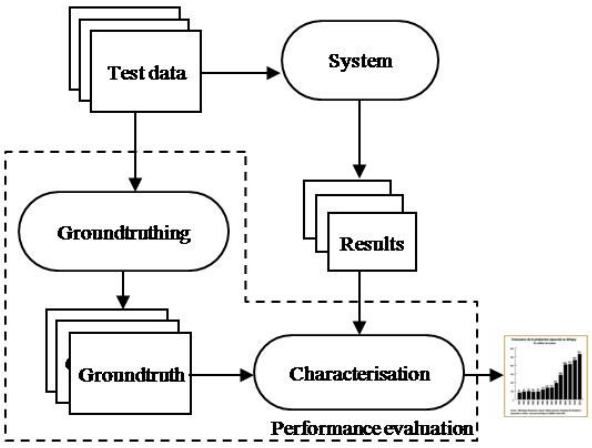


Fig. 1 Performance evaluation

In this paper we are especially interested in the first of these two tasks, applied to two particular topics of DIA: symbol recognition & spotting. Figure 2 compares these two processes. Symbol recognition is an active topic in the field of graphics recognition. Several surveys [5–8] review the existing work on logical diagrams, engineering drawings, maps, etc. In a very general way, a symbol has been defined as "*a graphical entity with a particular meaning in the context of an specific application domain*" and then, symbol recognition as "*a particular application of the general problem of pattern recognition, in which unknown input patterns are classified as belonging to one of the relevant classes (i.e. predefined symbols) in the application domain*" [7].

[1] Content Based Image Retrieval
[2] Document Image Analysis

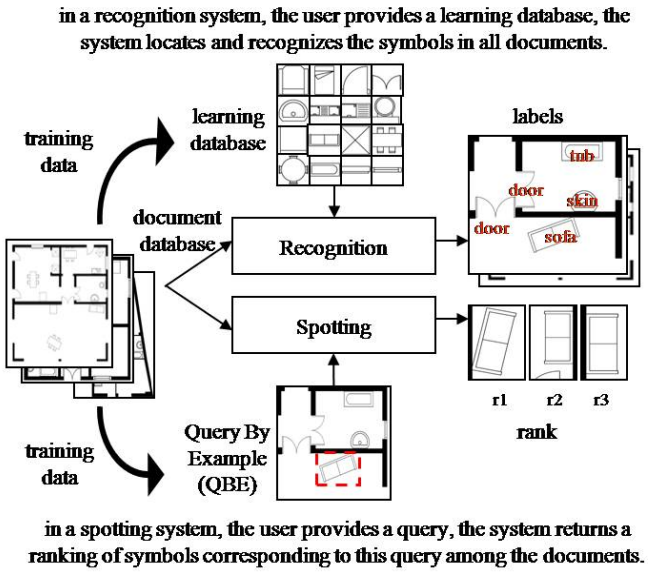


Fig. 2 Recognition/spotting of symbols

Usually, symbols do not appear isolated, but are connected to other elements of the document (connecting lines, background, other symbols, etc.). Thus, one of the major problems of symbol recognition is to combine segmentation and recognition. This problem is known as the segmentation/recognition paradigm in the literature [9]: a system should segment the symbols before recognizing them but, at the same time, some kind of recognition may be necessary to obtain a correct segmentation. In order to overcome this paradox, research has been directed to symbol spotting [10]. Since research on symbol spotting is just starting, definitions of symbol spotting are still a little ambiguous. In [8] it is defined as "*a way to efficiently localize possible symbols and limit the computational complexity, without using full recognition methods*". So, spotting is presented as a kind of middle-line technique combining recognition and segmentation. While symbol recognition tries to find the location and label of every symbol in the document, symbol spotting methods can be viewed as a kind of retrieval system [11–16]. Spotting is usually initiated with a query selected by the user from a drawing, what we call a QBE[3]. Then, the example is used as a model to find similar symbols in the document database. At the end, the system provides a ranked list of similar symbols along with their localization data (i.e. url of the source document with the coordinates of the symbol).

In both cases (spotting and recognition), a hard problem is how to obtain and compare experimental results from existing systems. Traditionally, this step was done independently for every system [5–8], by comparing manually the results with the original images and checking the recognition errors. This process was unreliable as it raises conflicts of interest and does not provide relevant results. Moreover, it does not allow the comparison of different systems or support testing with large amounts of data. In order to solve these problems, research has been initiated over the last few years on the performance evaluation of symbol recognition/spotting systems [17], resulting in the organization of several international contests on symbol recognition [18–21]. However, this work has focused on the recognition of isolated symbols. They do not take into account segmentation or spotting of symbols in real documents. The main reason for that is the difficulty of obtaining a large set of documents with the corresponding groundtruth. Doing that manually would require an unaffordable amount of time, as all the symbols in the document must be precisely located and labeled.

In this paper we propose a new approach to the automated generation of test documents to enable performance evaluation of symbol recognition and/or spotting systems in real context. The novelty in this approach is that it is constraint-driven and general enough to be applied to different domains. The constraints permit us to place the symbols on predefined backgrounds. They are defined according to a particular domain such as electronics, architecture or engineering. Thus, by simply defining the set of constraints and providing a few predefined backgrounds, we are able to produce a large amount of images resembling to real documents. As documents are synthetically generated, the groundtruth corresponding to the location and the label of every symbol becomes automatically available. We have applied this approach to the generation of large databases of architectural drawings and electronic diagrams, which shows the flexibility of the system.

In the rest of the paper, firstly we will review in section 2 the previous work done on groundtruthing applied to symbol recognition & spotting. In section 3, we will present our approach to generate synthetic documents based on positioning constraints of symbols. In section 4, we will introduce the graphical user interface to define the constraints and generate the documents. The results of the application of the system, to the generation of architectural drawings and electronic diagrams, are presented in section 5. Finally, in section 6 we state the main conclusions and future perspectives of this work.

2 Overview

The first step to evaluate any graphics recognition system is to provide test documents with their corresponding groundtruth data [22]. Concerning the specific topic of symbol recognition & spotting, several systems have been proposed in recent years: [8], [23], [24], [18], [25] and [17]. They can be classified in two main approaches: based on real or synthetic data. In the first case, test data consists of real documents. The groundtruth is defined from these documents by human operators using

[3] Query by Example

GUI[4]. Concerning synthetic data, the documents are built automatically, or semi-automatically, with their corresponding groundtruth by the systems, using some models of document generation.

In the rest of this section, we will present and discuss both approaches in subsections 2.1 and 2.2. In addition to that, to support this discussion we compare the systems in Table 1 according to different criteria: speed of the groundtruthing process, realism of test documents, reliability, number of symbols per image, generation of connected or disconnected symbols and ability to add noise with degradation methods. Finally, conclusions are drawn in subsection 2.3.

-- weak ++ good	Speed	Realism	Reliability	Symbol per document	Symbol connected	Degradation	
Tombre'06 [8]	--	++	-	many	yes	no	real data
Yan'04 [23]	-	++	--	many	yes	no	
Chhabra'98 [24]	+	++	+	many	yes	no	
Aksoy'00 [18]	++	--	++	many	no	yes	synthetic data
Zhai'03 [25]	++	--	++	one	no	yes	
Valveny'07 [17]	++	--	++	one	no	yes	

Table 1 Comparison of groundtruthing systems

2.1 Real Data

The natural approach to obtain the groundtruth is to define it from real-life documents. In that case, GUI are used by human operators in order to edit manually the groundtruth data. Thus, the groundtruthing starts from raster images in order to provide a vector graphics description of the content (e.g. graphical labels, region of interest, etc.). As the groundtruth data is edited by humans, it is necessary to do this task collaboratively with different operators [22]. In this way, errors produced by a single operator can be avoided.

In the past, this approach has been mainly applied to the evaluation of layout analysis and OCR[5] systems [26–28]. Concerning symbol recognition & spotting, only the EPEIRES[6] platform exists to date [8][7]. It is presented in Figure 3. This system is based on a collaborative approach using two main components: a GUI to edit the groundtruth data connected to an information system. The operators obtain, from the system, the images to annotate and the associated symbol models. Groundtruthing is performed by mapping (moving, rotating and scaling) transparent bounded models on the document using the GUI. The information system allows users to collaboratively validate the groundtruth data. Experts check the groundtruth data generated by the operator by emitting alerts in the case of errors.

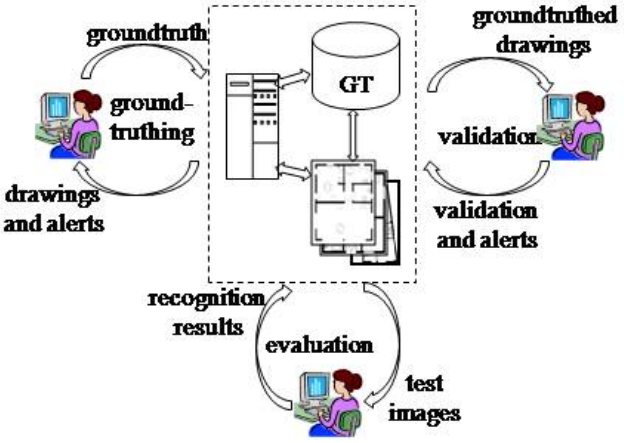


Fig. 3 The EPEIRES system

Despite this existing platform a problem still remains [22]: the time and cost required to edit the groundtruth data. Similar contributions made in OCR and layout analysis [26–28] highlight that, in most of the cases, the groundtruthing effort makes the creation of large databases very hard. An alternative approach to avoid this problem is semi-automatic groundtruthing. In this case, the key idea is to use a recognition method to obtain an initial version of the groundtruth data. Then, the user only has to validate and to correct the recognition results in order to provide the final groundtruth data. This approach has already been used in other applications like OCR [29], layout analysis [30], chart recognition [31], etc.

Concerning symbol recognition & spotting, only the system described in [23] has been proposed to date. This system recognizes engineering drawings using a case-based approach. It is mainly used to learning and recognition, but it could be easily extended to groundtruthing. The user starts by targeting a graphical object (i.e. a symbol) in an engineering drawing. The symbol is next vectorized into a set of straight lines and represented as a model tree. This model tree is used to localize and recognize similar objects in the drawing. During the learning process, the system also takes into account user feedback on positive and negative examples. It modifies the original tree by computing tolerances about the primitives and their relations (length, angle, line number, etc.).

In any case, the systems presented previously render the groundtruthing impractical for constructing large-scale databases. In [24], the authors propose an alter-

[4] Graphics User Interface(s)
[5] Optical Character Recognition
[6] http://epeires.loria.fr/
[7] None related publication exists about this platform, we refer the reader to the description given in [8].

native way to solve this problem. Their key idea is to use vector graphics documents (e.g. "DXF, SVG, CGM, etc."), and to convert them into images. In this way, they can take advantage of already existing groundtruth data: it is not necessary to re-define it. Their system has been used to evaluate raster to vector conversion [24]. However, it could be easily extended to symbol recognition & spotting by using the symbol layer of the CAD files. The remaining difficulty in this approach is to collect and to record the electronic documents [32]. Several problems still exist: to check the copyrights of documents, to organize the documents in the database (to define single id, to associate duplicates, etc.), to validate the formats and to convert them to a standard one if necessary, to edit metadata about the documents in order to index the database, etc.

2.2 Synthetic Data

The systems using real-life documents result in realistic test data but render the groundtruthing complex (errors, delay and cost, copyright, database indexing, etc). A complementary approach, which avoids these difficulties, is to create and to use synthetic documents. Here, the test documents are generated by an automatic system which combines pre-defined models of document components in a pseudo-random way. Test documents and groundtruth can therefore be produced simultaneously. In addition, a large number of documents can be generated easily and with limited user involvement. Several systems have been proposed in the literature [18] [25] and [17], mainly used in the context of international contests of symbol recognition. Figure 4 gives some examples of documents produced by these systems.

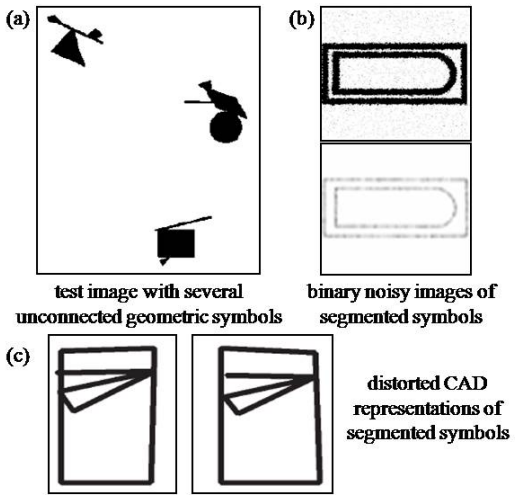


Fig. 4 Examples of synthetic documents [18] [25] [17] (a) random symbol sets (b) (c) distorted segmented symbols

The system described in [18] employs an approach to build documents composed of multiple unconnected symbols. Figure 4 (a) gives an example of such a document. Each symbol is composed of primitives (circles, lines, squares, etc.) randomly selected and mildly overlapped. Next, they are placed on the image at a random location and without overlapping with the bounding boxes of other symbols. The systems proposed in [25] and [17] support the generation of degraded images of segmented symbols as shown in Figures 4 (b) and (c). In these systems, the models of the symbols are described in a vector graphics format. They use a random selection process to select a model from the model database, and apply to it a set of transformations (rotations, scaling, and binary or vectorial distortions).

2.3 Conclusion

In recent years, several pieces of work have been undertaken to provide groundtruthed databases in order to evaluate symbol recognition & spotting methods, using real [8] [23] [24] as well as synthetic [18] [25] [17] data. As indicated in Table 1, the time needed to collect and groundtruth the real-life documents makes their use, for constructing large-scale databases, complex. Moreover, the groundtruthing is done by human operators making the results unreliable. For that reason, synthetic data have been mainly used to date for the evaluation of systems, for example during the international contests of symbol recognition [18,19,8,21]. With such data, test documents and groundtruth are produced simultaneously. Then, as mentioned in Table 1, data can be generated quickly making the production of statistically important groundtruth datasets feasible. Moreover, the groundtruth is produced directly from the models and, therefore, without errors. Finally, the content of documents can be controlled, which is an interesting property to evaluate the methods regarding scalability, geometry invariance, noise robustness, etc.

The major problem when using synthetic data is the difficulty of reproducing the variability of real documents. The systems proposed in the literature [18] [25] [17] only generate documents composed of segmented symbols, no whole documents which is the original goal of groundtruthing systems. Indeed, real-life documents (engineering and architectural drawings, electrical diagrams, etc.) are composed of multiple objects constrained by spatial relations (connectivity, adjacency, neighbourhood, etc.). Systems capable of generating whole synthetic documents would be very helpful. Such systems would provide a much more realistic context in which evaluation could take place. In this paper we present some contributions in this direction that we will introduce in next section 3.

3 Our Approach

In this paper we present a new approach to the building of synthetic documents for the performance evaluation of symbol recognition & spotting systems. Our key contribution is the building of whole documents (drawings, maps, diagrams, etc.) and the underlying aim of our work is to make the produced documents more realistic. The design of a suitable process is a challenging task. Indeed, realistic documents cannot be produced without importing human know-how into the process. In our work we have considered a shortcut way to solve this problem. Our key idea observes that graphical documents are composed of two layers: a background layer and a symbolic one. We use this property to build several document instances as shown in Figure 5. We generate several different symbolic layers and place them on the same background obtaining different documents. In this way, the building process is made easier and can be considered as a problem of symbol positioning on a given document background.

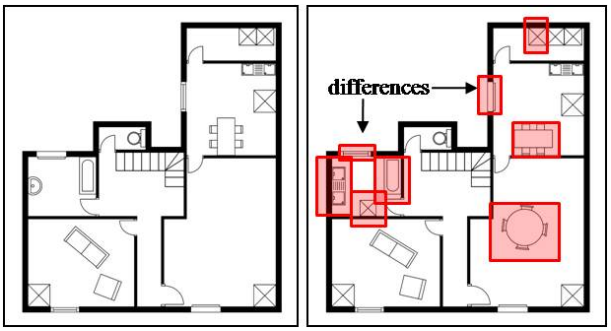


Fig. 5 Two document instances

In order to place randomly symbols on a given background, we have developed a building system based on the use of positioning constraints. These positioning constraints determine where and how the symbols can be placed on a background image. Figure 6 presents the architecture of our system. It uses as input data a background image, a database of symbol models and a file describing the positioning constraints. Using these data, it generates vector graphics documents with the corresponding groundtruth. These documents are next rasterized and noise added, to generate the final test images for evaluation. The central part of our system is the building of documents. Two main processes take part: symbol positioning and document generation. Symbol positioning is in charge of placing symbols on the background according to the parameters and conditions of given constraints. Document generation controls the positioning process, through a building loop including upstream and downstream steps, to ensure the generation of correct documents. It starts with empty documents and fills them with symbols in a pseudo-random way. The interactions between the two processes are explained in the workflow diagram presented in Figure 7. For each loop, a constraint and a symbol are selected. Then, the symbol is placed on the background according to the specifications of the constraint. After that, several tests are carried out to ensure that the symbol is well positioned. The process continues until some stopping criteria (concerning the number of symbols and the amount of free space) are satisfied.

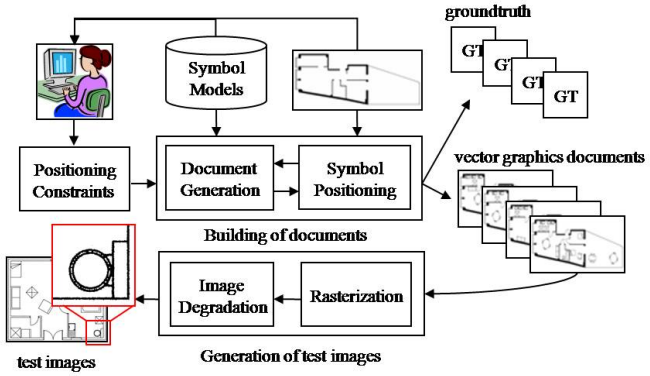


Fig. 6 System overview

In the rest of this section we will introduce first in subsection 3.1 the positioning constraints. Next, in subsections 3.2 to 3.8 we will describe all the steps (1) to (7) mentioned in the workflow diagram of Figure 7, and will detail how the constraints are processed at each of these steps. The last subsection 3.9 will give details about the final test image generation process.

3.1 Positioning Constraints

The key mechanism employed in our system is symbol positioning using constraints. Figures 8 (a) and (b) explain how it works. The constraints specify which symbols can be placed and where in a background (Figure 8 (a)). Each constraint defines a set of symbol models to instantiate, a shape to specify where symbols can be placed on the background (either a single point, a straight-line or a polygon), and some parameters that specify how these symbols are placed (concerning geometric transformations and the definition of control points). Then, a symbol is placed in such a way that its control point will match a point inside the shape defined in the constraint (Figure 8 (b)). More specifically, a constraint addresses the following issues:

Symbol models: The list of symbol models that can be selected to instantiate symbols to place.

Constraint size: The maximum number of symbols that can be placed using this constraint.

Constraint satisfaction: It specifies whether the constraint is mandatory (at least one instance of a symbol must be placed using this constraint) or not.

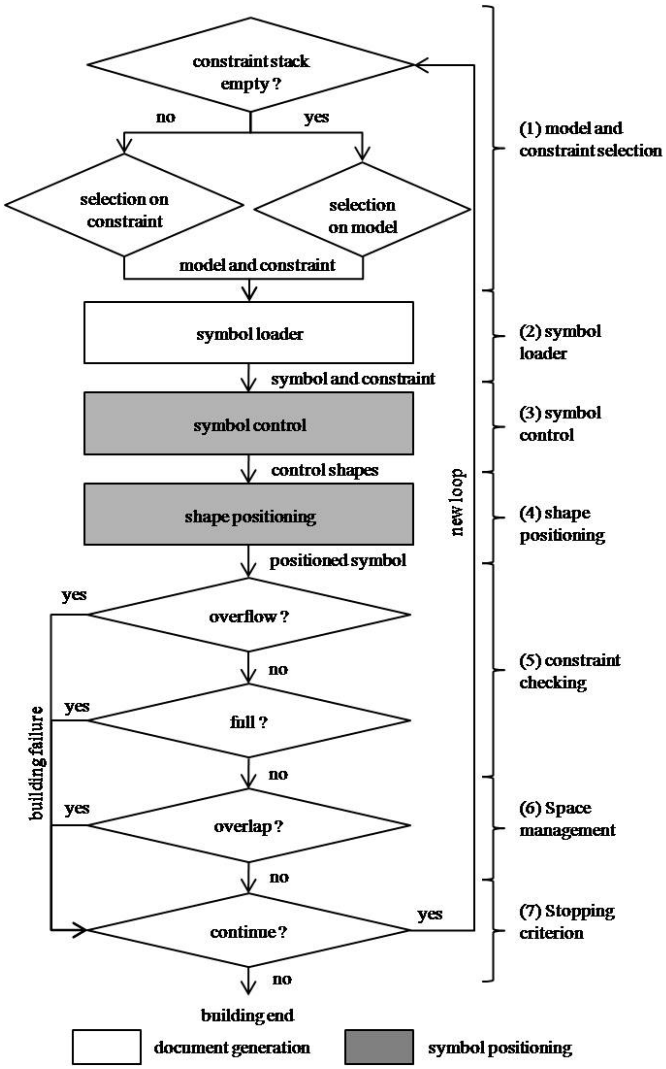


Fig. 7 Building workflow

Geometric transforms: Geometric transformations (rotation, scaling) along their associated parameters to apply to the symbols before placing them on the background.

Positioning shape: It defines where the symbols can be placed on the background. It can be a single point, a straight-line or a polygon. In the two last cases, the exact locations of symbols will be selected at random within the straight-line or the polygon.

Positioning control: The parameters to compute the control points used to position the symbols within the shape.

Figure 9 gives an example of a rule declaration used in the positioning. It is composed of a model and an associated constraint. This specifies mainly the link(s) between the model and the constraint and the parameters used for the positioning (the control point and the shape coordinates). Additional parameters can be employed to produce more complex rules. Table 2 gives the full explicit list of parameters we use in our rules. These

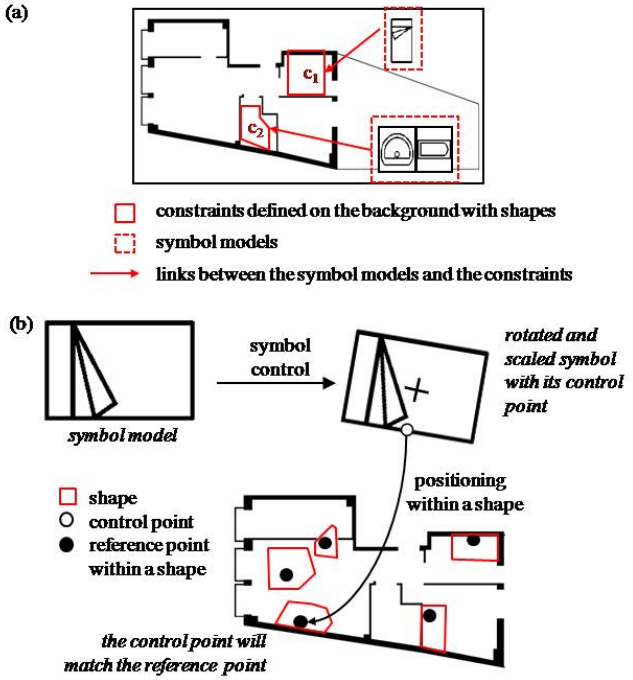


Fig. 8 Positioning constraint
(a) model/constraint link (b) positioning mechanism

are employed at different steps (1) to (7) of our building process shown in Figure 7. We will detail each of them in next subsections.

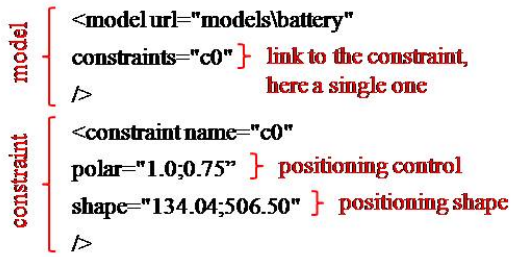


Fig. 9 Rule declaration

3.2 Constraint & Model Selection

To initiate a building loop in our workflow (Figure 7), we have to select a symbol model and an associated constraint. We have implemented two selection modes: a constraint-based selection and model-based selection. The motivation for defining two selection strategies is that some constraints are mandatory to be satisfied while others are optional, as illustrated in Figure 10. For instance, doors and windows are mandatory in order to close the house and a bed is also required in a bedroom while a sofa is optional. Constraint-based selection permits to ensure that all mandatory constraints are taken

	parameter	default value	description
model	url	required	url of the model file
	constraints	required	constraint(s) linked to the model
	morph	1.0	to control the thickness of symbol lines
constraint	name	required	name of the constraint
	rigid	true	to specify if the constraint is mandatory
	scale	1.0	to scale the symbols
	rotation	0.0	to rotate the symbols
	polar	{0.0;0.0}	parameters to compute the control point
	align	0.0	parameter used for the rotation alignment
	shape	required	coordinates of the shape
	overlap	false	the symbol can overlap the other ones
	overflow	false	to coerce the positioning within the shape
	size	1	maximum number of symbol to place

Table 2 Rule parameters

into account in the generation of the document, while model-based selection also includes optional constraints.

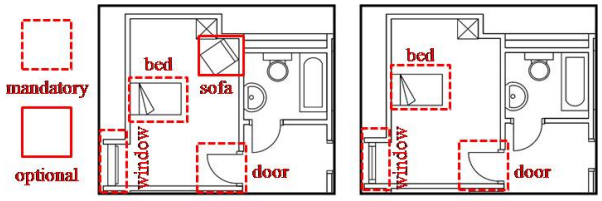


Fig. 10 Constraint satisfaction

The constraint based selection selects the constraints first and the symbol models next. This way, it will coerce the positioning i.e. the constraint will be satisfied. Figure 11 (a) details the selection process we use. It is done by managing a constraint stack. All the constraints defined as "mandatory" in the constraint file are loaded and pushed into this stack. Then, when starting a building process these constraints $(c_1, ..., c_i)$ are popped-up from the stack at each loop. A symbol model (s_k) linked to this constraint is next selected at random using a uniform probability distribution $(p_1, ..., p_j)$. When the stack is empty the document generation shifts to the model based selection as illustrated in Figure 7.

The model based selection works in the opposite way. It selects symbol models first and linked constraints next. The selection of symbol models is performed so that it will satisfy a maximum number of constraints among documents. We favor symbols linked to several constraints, or those linked to weak associated constraints. The constraints are next selected at random

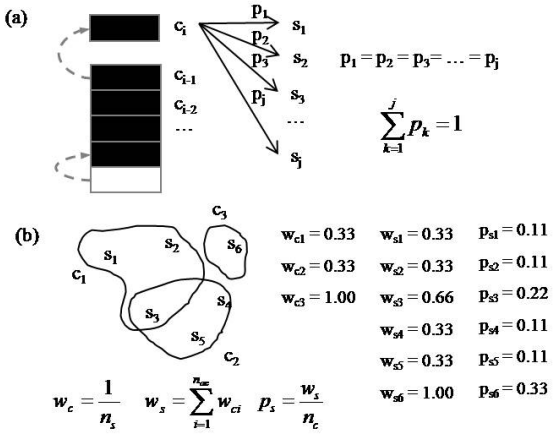


Fig. 11 (a) constraint selection (b) model selection

with an uniform probability distribution. This guaranties a better spatial distribution of symbols among documents, and thus a better visual rendering of documents. Figure 11 (b) presents the method we use to achieve this. First, a weight w_c is computed for each constraint based on the number of symbols n_s associated with it. The weight w_s of a symbol is next obtained by summing the weights w_c of constraints linked with it. The obtained w_s are at last normalized by the total number of constraints n_c to obtain the selection probabilities p_s of symbols. Using these probabilities, a symbol model can be selected at random. This way, the system increases the selection probabilities of symbols linked to several constraints, or those linked to weak associated constraints, like respectively s_3 and s_6 in Figure 11 (b).

3.3 Symbol Loader

Once a model is selected, we instantiate the corresponding symbol by loading its model file. These model files are kept inside our database as illustrated in Figure 6. They are in a vector graphics format, describing the symbols using geometrical primitives (straight lines, arcs and circles) with their associated thickness attributes. Once loaded, the symbols have to be adapted to the background image. Indeed, these background images are made from real-life document images (web images, digitized documents, etc.) picked up at random. As they are not adapted to our symbol library, we must apply a set of geometric transformations (scaling, morphing and random rotation) to the symbols before placing them on the background.

Scaling aims to adapt the symbols to the size of the background image. It employs a single parameter defined in the positioning constraint, to scale the symbols in relation to their gravity centers. In the same manner, the morphing operation adapts the thickness of the symbols to the background image. Figure 12 (a) gives two examples of a symbol placed on a background, without and

with thickness adaptation. In a last step, the symbols are rotated. This rotation is done using a parameter that can be null, a fixed value or a range. In the last case, the final value is selected at random inside the range. A gap can also be defined to sample the rotation values within the range. The key objective of making these rotations random, is to increase the variability of the documents. Figure 12 (b) gives an example of a tub rotated in two different ways using a same constraint. In this example, the range is $\{\frac{\pi}{2}, \frac{3\times\pi}{2}\}$ with a π gap.

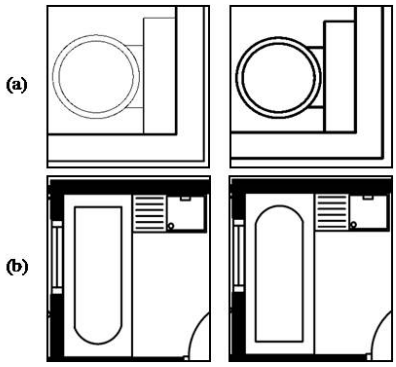


Fig. 12 Geometric transforms (a) morphing (b) rotation

3.4 Symbol Control

Once a symbol is loaded our system initiates the positioning. The first step is to compute the control point of the symbol as detailed in Figure 13. We define this control point in relation to the bounding box of the symbol. Indeed, the bounding box is a common way to handle graphical objects inside a document analysis system. The method we employ to compute it is given in Annex A1. It takes into account the thickness of the lines and permits also to apply an alignment rotation, both to the symbol and the control point. It allows the symbol to be aligned to the background elements. Figure 13 gives an example of a symbol (*a sofa*) aligned to a background element (*a wall*).

In our approach, we have made the computation of control points fully independent of the symbol models. Like this, we make the association of different models to a single constraint easier i.e. it is not necessary for a user to define a specific control point for each of the models. For that purpose, we have defined the control points in our constraints with polar coordinates as shown in Figure 14. The key process is then, starting from a control point p defined in the polar space, to find for a given symbol $_i$ the right control point p_i in its bounding box. In the polar space the control points p are represented using two coordinates (L, θ). We use then some standard geometric methods to find the right length and direction (L_i, θ_i) for a given symbol (see Annex A2).

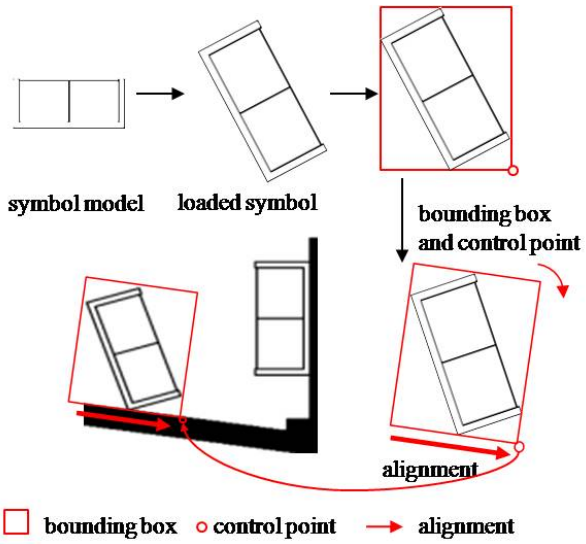


Fig. 13 Symbol control

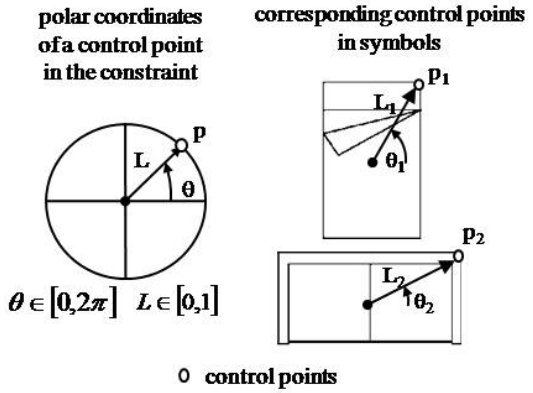


Fig. 14 Definition of control points

3.5 Shape Positioning

Once the control point is computed, we position the symbol on the background. It is based on the matching of the control point with a positioning one defined on the background. The symbol will be then positioned so that its control point matches with the positioning one. In order to introduce some variability in the built documents, we employ different possibilities to select the positioning point in a constraint. It can be defined as being a fixed point (x, y) or can be selected at random inside a straight line (the point is selected at random along the line) or a polygon (the point is selected at random inside the polygon). Figure 15 (a) and (b) gives examples of random positioning in both cases. To perform the random selection, we employ different computational geometry methods detailed in Annex A3.

3.6 Checking of Constraints

When a straight-line or a polygon is used to select the positioning point, their boundaries could also be employed

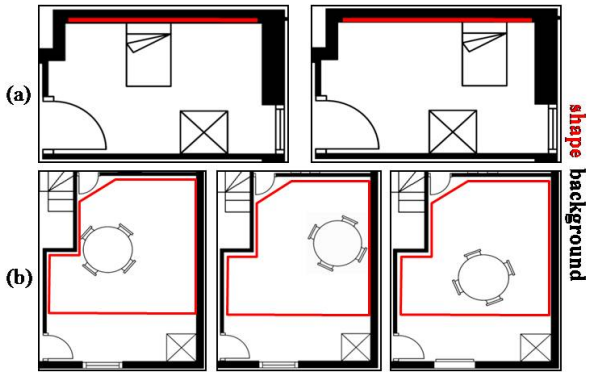


Fig. 15 Random positioning (a) straight line (b) polygon

as delimiters to constraint the positioning of the whole symbol. Figures 16 (a) (b) present some examples where this option could be useful. In the case of Figure 16 (a), the straight-line delimits the positioning of the sofa to a wall part. In this way, a sofa will not be placed in front of the room entrances. The case of Figure 16 (b) shows how a polygon could constrain the positioning of a table within a hall. Then, this table will not obstruct the way to the window or be placed along the flat's wall.

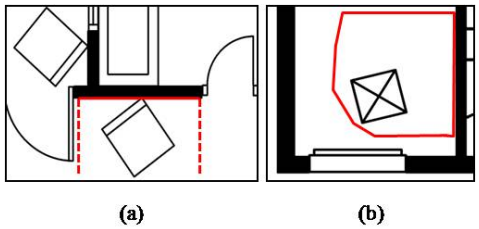


Fig. 16 (a) line delimiter (b) polygon delimiter

In our system, we allow the user to choose if a constraint has to be used as delimiter or not. Such a choice will depend on the context of the constraint and the user will decide then in regard to his domain know-how. However positioning might fail. Such failures appear when parts of a symbol overflow the area of the constraint. To check it, we use overflow tests between the symbols and the straight-line/polygon shapes defined in the constraints (Figure 17). Both are based on some standard overlapping and line-intersection methods as detailed in Annex A4. A positive result will cancel the positioning. Afterwards the document generation will initiate a new building loop.

Another particularity related to the use of straight-lines or polygons in the constraint, is that more than one symbol could be positioned. Figure 18 gives some examples of that. In the case of Figure 18 (a) several sinks are placed along a wall using a straight-line. Concerning the case of Figure 18 (b), the furniture composing a living-room is placed at random inside a polygon. Thus, in our approach we have allowed the user to specify a maximum number (between 1 to n) of symbols to position per constraint. When a constraint becomes "full", the positioning is canceled.

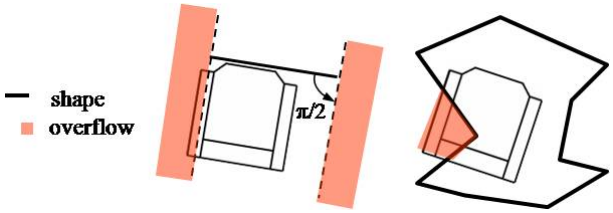


Fig. 17 Overflow of symbols

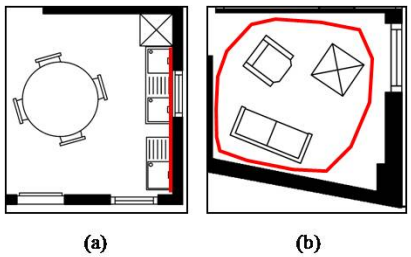


Fig. 18 multiple positioning (a) straight-line (b) polygon

3.7 Space Management

In addition to the checking, the system also continuously monitors the document space. Indeed, during the building process the document is filled gradually. The system has to ensure that any new generated symbol falls entirely within the image and do not overlap with the existing symbols. Such detection is achieved using overlapping tests (see Annex B5) between the bounding box of the new symbol and all the already positioned in the document.

However, it could appear than some symbols have to be mildly overlapped. This case appears when we want to make connect some symbols on the background. Figure 19 (a) gives examples of that in a bathroom and in a cellar. In order to take into account these cases in our system, we work with two building layers as illustrated in Figure 19 (b). In the first layer we forbid the overlapping as detailed previously. In the second layer we allow a mildly overlapping between symbols. We use the line width of symbols as a threshold to control the overlapping degree. At the end, the symbols placed in this second layer are overlayed to the first layer and the background, to produce the final document. The selection of a given layer for a constraint is set up by the user. Like this, the mildly overlapping between symbols will be allowed according to his needs.

3.8 Stopping Criterion

Our building system starts with empty documents and fills them in a pseudo-random way with generated sym-

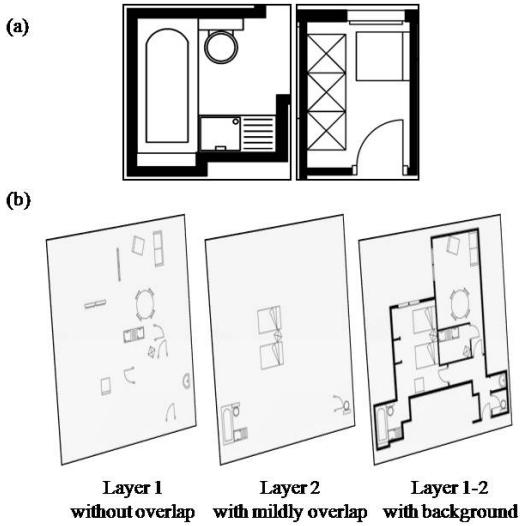


Fig. 19 (a) side by side symbols (b) building layers

bols. The building process is stopped when the maximum number of symbols to position in the document is reached. This number corresponds to the sum of the maximum numbers of allowed symbols per constraint. However, in some cases a complete satisfaction of all constraints could be difficult to achieve. The user could define a large number of symbols per document, that will be hard to achieve without relaxing the defined constraints. The system must then detect and count these cases, and stop the building if necessary in order to avoid an infinite building process.

To achieve this, we use the checking and space management tests presented in subsections 3.6 and 3.7. When the results of these tests are negative, we trigger a building failure as shown in Figure 7. We count then these building failures and compare them to a threshold set up by the user. He defines it in relation to the edited constraints, the considered domain and background image, his satisfaction requirement, etc. If the number of building failures becomes greater than this threshold, we stop the process.

3.9 Generation of Test Images

Once vector graphics are documents obtained, we convert them into raster images for performance evaluation. However, to test recognition systems one needs noisy images. In our system, we use two different workflows to add noise to the images (Figure 20): scan-based and web-based. The first one aims to distort the images in a way similar to the scanning process, whereas the second produces low-resolution and lossy compressed images as the ones found on the Web.

We exploit three processing steps to produce the scan-based images: scaling, rasterization and image degradation. Vector graphics documents are first scaled

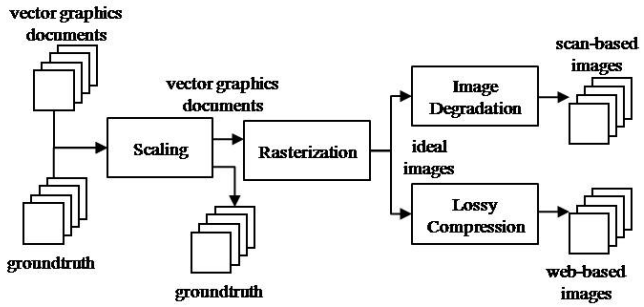


Fig. 20 Generation of test images

with their corresponding groundtruth. Indeed, our synthetic documents are produced from pre-defined background images. These are selected at random from digital libraries and therefore appear at different scales. Thus, we resize all the produced documents in order to put them at a same scale in our datasets. The groundtruth data are also resized in order to keep them valid. Next, we rasterize the vector graphics documents using tools such as ImageMagick[8] or Inkscape[9]. Rasterized images are obtained in gray-level, we binarize them using a fixed threshold. In a final step we employ the image degradation algorithm of [33]. This algorithm tries to reproduce the process of printing and acquisition. It has been used in different applications of DIA (especially OCR), and in all the past contests on symbol recognition [18–21]. Figure 21 (a) gives examples of degraded images using this algorithm.

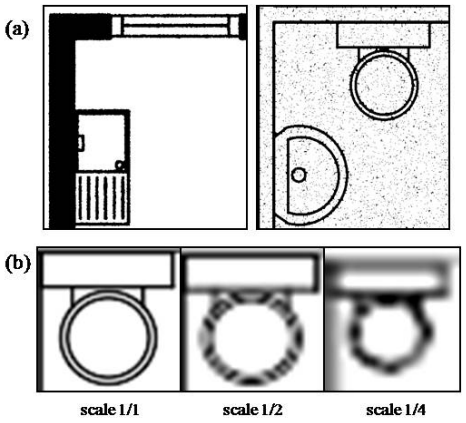


Fig. 21 Image degradation
(a) Kanungo (b) Low resolution

Web-based images are produced using a similar workflow. Vector graphics documents are also scaled with their corresponding groundtruth to make them homogenous, and are next rasterized. However, they are scaled at lowest levels to obtain low resolution images as the ones found in the Web. Figure 21 (b) gives some exam-

[8] http://www.imagemagick.org/
[9] http://www.inkscape.org/

ples of degradation at lowest resolutions (from scales 1/1 to 1/4). In a final step we compress the rasterized images with the jpeg lossy compression algorithm.

4 Graphics User Interface

In previous section 3 we have presented our system to generate synthetic documents and their corresponding groundtruth at the symbol-level. The system relies on the use of positioning constraints. It employs different entry data, a background image, a set of symbol models and a file describing the positioning constraints. As presented previously, the positioning process exploits various operations (scaling, morphing, random and alignment rotation, etc.). Therefore, a constraint file is expected to contain a lot of data depending on the number and the complexity of constraints. Thus, it can be difficult to edit it manually.

In order to help the user, we have developed a GUI allowing different editing tasks: loading and attaching a background image, loading a model database, creating and placing constraints, linking constraints to models, setting constraints, saving and loading constraint files, etc. Figure 22 gives a snapshoot of this GUI with some edited constraints. In practice it is difficult to edit the constraints without observing their effect on document building. In order to make the editing easier, we have plugged our GUI to our building engine as shown in Figure 23 (a). Using this plugin, the editing of constraints is done in interaction with the user in three steps: (1) editing process in progress (2) running of the engine (3) display of the building result. This last step relies on a building viewer presented in Figure 23 (b).

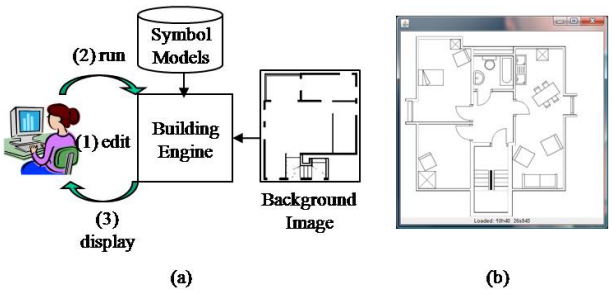


Fig. 23 (a) GUI/Engine plugging (b) building viewer

An editing process could take 1 to 3 hours per background, depending on the number and complexity of constraints (from 20-40). A user must be familiar with the document domain (architectural, electrical, etc.). In addition, he must be trained about the constraint mechanism employed in our system, that involves some skills in computer science. Thus, this system is mainly intended for people working on the computer vision field, interested in the performance evaluation aspects. At this time, it has been employed by different Master and PhD Students following a short training (half a day).

Once all the constraints are edited from a background image, the user can produce a full database of synthetic documents using the main GUI. It is only necessary to specify the total number of documents to be generated and stored in the database. The system will export the test documents to a directory with the corresponding groundtruth files. Groundtruth files contain metadata about the symbols composing the test documents: locations of bounding boxes, labels, orientations, scaling and morphing parameters. Annex D gives an example of the content of groundtruth files. Test documents are in a vector graphics format (i.e. with vectorial data for the symbol layer and a raster image for the background). In a final step, we use batch processing to rasterize the vector graphics documents and to add noise to the obtained images, as explained in subsection 3.9.

5 Experiments and Results

In this section we present the experiments performed and the results obtained with our system. The main objective of these experiments is to create collections of test documents, with the corresponding groundtruth, to be used in evaluation frameworks of symbol recognition & spotting systems. To achieve this, we set up our system so that it generates different collections of test documents. Our key objective is to highlight the flexibility of our approach[10]. Table 3 gives the details about these collections in terms of the numbers of datasets, images, symbols placed on the documents and models used. All these collections are free and downloadable from our website[11]. In next subsections 5.1 to 5.3, we will present the collections of documents we have produced, and how we have set up our system to do it. In subsection 5.3, we will highlight how our approach is suitable for performance evaluation, by presenting characterization results of a spotting system obtained on our document collections.

Collections	Datasets	Images	Symbols	Models
bags	16	1600	15046	25-150
floorplans	10	1000	28065	16
diagrams	10	1000	14100	21
	36	**3600**	**57211**	

Table 3 Collections of test documents

5.1 Bags of Symbols

We have built a first collection composed of "bag of symbols" documents. Figure 24 presents some examples of

[10] Our system has been also used to produce geographic maps to evaluate text/graphics separation algorithms.
[11] http://mathieu.delalandre.free.fr/projects/sesyd/

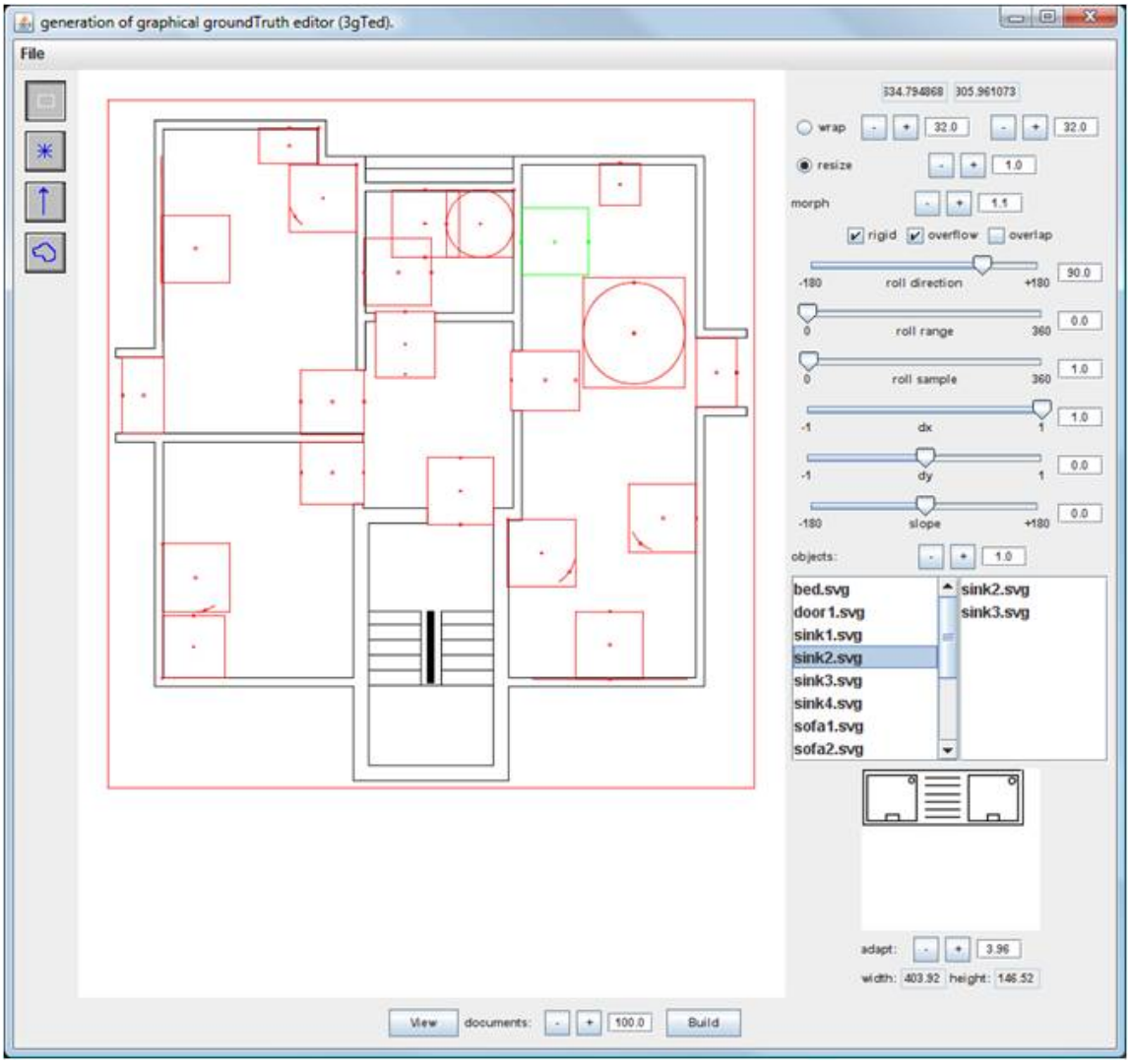


Fig. 22 GUI to edit the positioning constraints

these bags. In them, the symbols are positioned at random on an empty background, without any connection, and using different rotation or scaling parameters. These bags present an "easy" localization problem. The key idea of this collection is to establish a bridge with the datasets provided during the past contests on symbol recognition held during the GREC[12] Workshop [19, 8, 21], composed of only one segmented symbol per image.

To produce this collection, we have used the symbol model library[13] created during the previous editions of the GREC contest. This library is composed of 150 models of architectural and electrical symbols. Based on this library, we have set up our system with a single square zone constraint surrounding an empty background. In order to produce bags of a reasonable size, we have resized the original symbol models of the past contest editions[13] from 512×512 to 256×256 pixels. Based on this symbol size, we have generated bags of 1024×1024 pixels composed of around 10 symbols each. This corresponds to a symbol density of 0.625 ($\frac{256^2 \times 10}{1024^2}$),

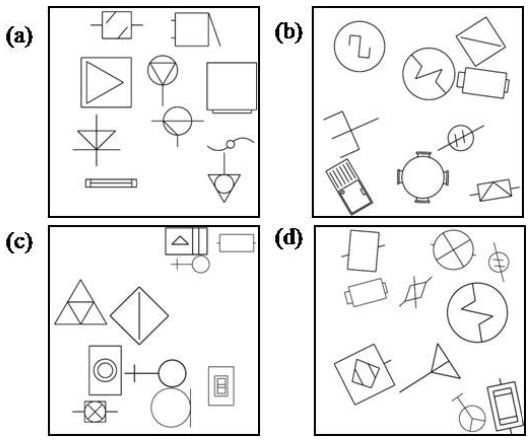


Fig. 24 Examples of bag of symbols
(a) none transformation (b) rotated
(c) scaled (d) rotated & scaled

[12] Graphics Recognition
[13] http://epeires.loria.fr/

which respects a well-balanced partitioning between the background and the foreground parts as shown in Figure 24.

Using these size parameters we have generated 16 datasets of 100 bags each. This corresponds to an overall number of 1600 bags composed of around 15000 symbols as indicated in Table 3. These 16 datasets have been generated by respecting the protocol used during the previous runs of the contest[13]. First we have used different model numbers (25, 50, 100 and 150) in order to test the scalability of the methods. Next we have applied and combined different geometrical operations as illustrated in Figures 24 (a), (b), (c) and (d). These transformations have been set up as follows: from 0 to $2 \times \pi$ for the random rotation with a gap of $\frac{2\times\pi}{1000}$, and from 75 % to 125% for the scaling with a gap of 0.05 % ($\frac{50}{1000}$).

5.2 Architectural Floorplans

Our second collection aims to provide whole documents using filled backgrounds. For that purpose, we have dedicated this collection to architectural floorplans. We have chosen architectural floorplans in recognition of their interesting properties concerning the connectivity and the orientation of symbols. Figure 25 presents some examples of floorplans automatically produced by our system.

In order to generate these floorplans, we have defined first a set of architectural symbol models. Our set is composed of 16 models, Figure 26 gives their thumbviews and labels. In this set, the sizes of the symbols respects existing proportions appearing in real-life floorplans. We give in Figure 26 the scales of thumbviews[14] regarding the real sizes of symbols in floorplans.

Then, we have used these models and some background images in order to produce our constraints. Figure 27 (a) explains the process we use. First, to create our backgrounds we have picked-up at random some images of real floorplans (digitized documents, web images, etc.). We have next cleaned these images by deleting elements like texts, symbols, arrows, dimensions, etc. Once the background images are obtained, we use our GUI to edit the constraints and to link them to the symbol models. As we want the documents to be as real as possible, we use during the editing step, the original images in order to reproduce the domain rules in the constraints. Figure 27 (b) gives some examples of the original floorplans, and the documents we have built from them. The initial information concerning the types and the locations of symbols has been preserved in the constraints.

We have created our whole floorplan collection using 10 different backgrounds. However, to address the time complexity problem of symbol recognition & spotting processes, we restrict ourselves to background images composed of a small number of rooms. Like this, we produce floorplan images of reasonable dimensions. The number of edited constraints per background image changes from 21 to 41. Using these constraints, we have generated 100 instances of documents per background. Our final collection of documents is composed of 1000 images and around 28 000 symbols to locate as indicated in Table 3. This corresponds to a mean number of 28 symbols per image, with a minimum of 18 and a maximum of 40. In a final step, we have scaled all the produced images to make their resolution homogenous across the whole database. The scaling parameter has been defined in order to reach a symbol size of 192×192 for the smallest symbol models and 460×460 for the

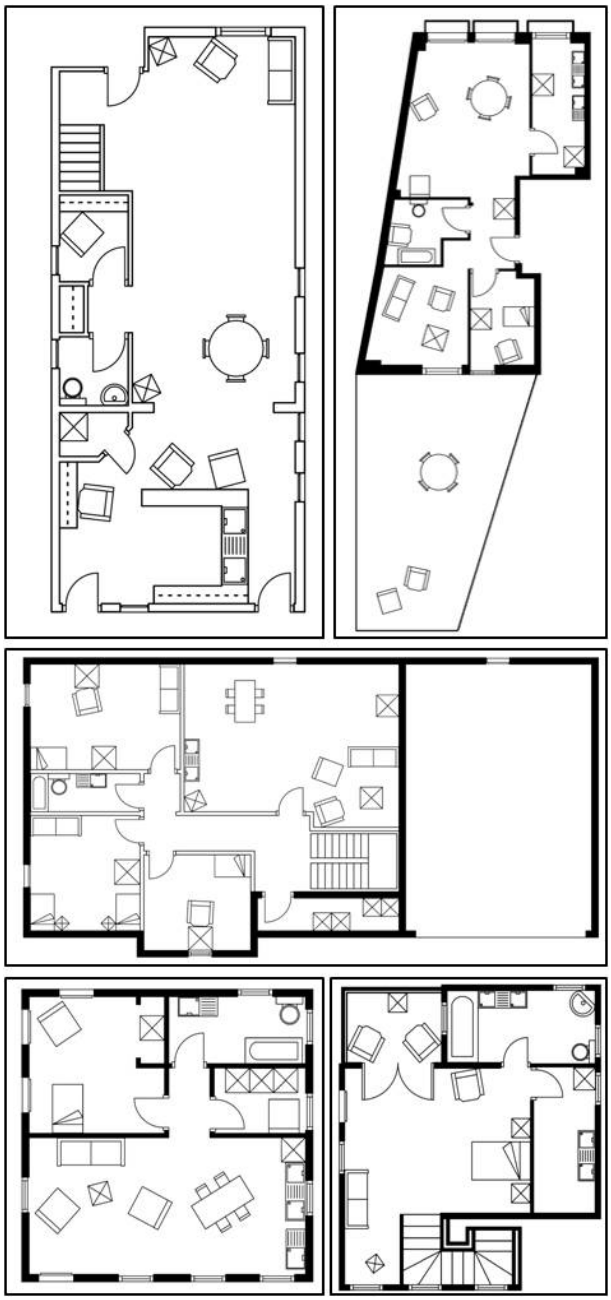


Fig. 25 Examples of built floorplans

[14] Ratios of thumbview sizes to corresponding symbol sizes in floorplans.

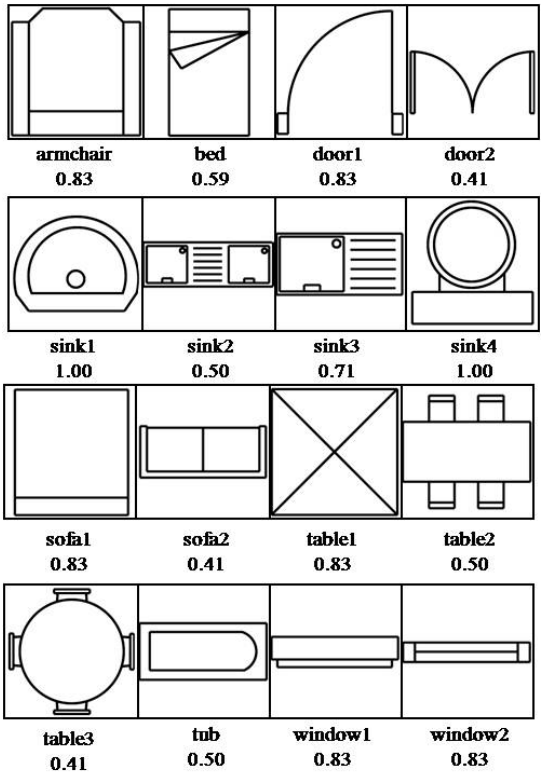


Fig. 26 Thumbviews of architectural models labels with scales[14]

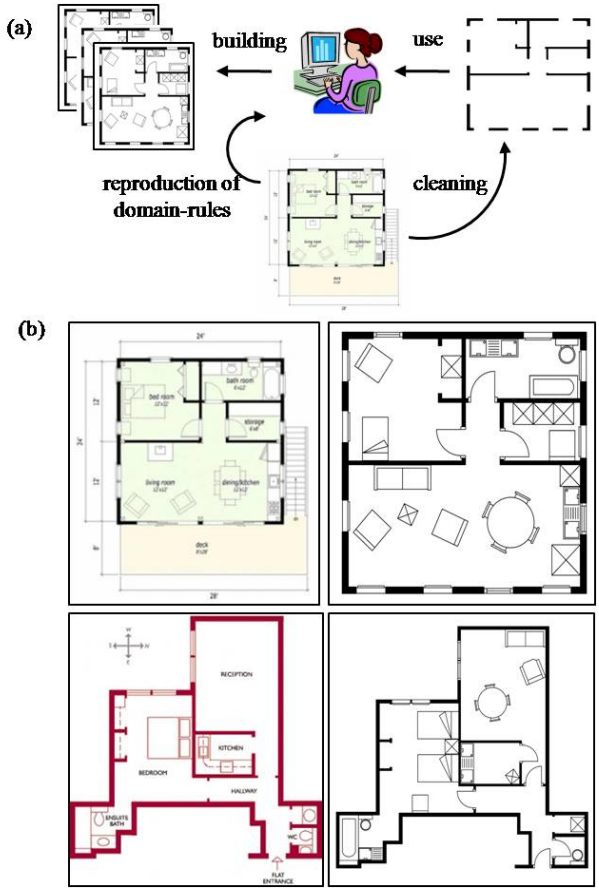


Fig. 27 (a) edition process (b) real-life vs. built documents

biggest ones (corresponding respectively to the scaling parameters 1.0 and 2.4 of Figure 26).

5.3 Electrical Diagrams

Our last collection is concerned with electrical diagrams. Our key objective here is to show that our approach is not domain dependent, and could be applied therefore to build other document types. Figure 28 shows some examples of diagrams we produce.

To build these diagrams we have first created a model library of electrical symbols. Figure 29 gives thumbviews of them, our set is composed of 21 models. Then, the process we have used to construct this collection is similar to the one of floorplans. In a first step, we have picked-up at random some images of real diagrams and cleaned them to obtain the backgrounds. The obtained background images contain only wires joining empty symbol places. Next, we have used our GUI to edit constraints for the obtained backgrounds by reproducing domain rules of original diagrams.

We have generated our whole collection of diagrams from 10 different backgrounds. We restrict ourselves to diagrams composed of a few number of components, in order to address the time complexity problem of the symbol recognition & spotting processes. Then, in the case of diagram collection the number of edited constraints per background image varies from 7 to 26. Using these constraints, we have generated 100 instances of documents per background. Our final collection of documents is composed of 1000 images and around 14 000 symbols to locate as indicated in Table 3. This corresponds to a mean number of 14 symbols per image, with a minimum of 7 and a maximum of 26. In a final step, we have scaled all the produced images to make their resolution homogenous across the whole database. The diagrams have been produced in order to respect a mean symbol size of 192×192 (from 57×57 for the smallest ones to 288×288 for the biggest ones).

5.4 Application to Performance Evaluation

In past subsections 5.2 and 5.3, we have illustrated how our system can produce documents that look realistic, by reproducing domain rules of true-life documents in positioning constraints. Using these constraints our system generates document instances i.e. different symbolic layers on a same background. In this subsection, we illustrate how these documents are suitable for performance evaluation. To do it, we have employed them to evaluate the symbol localization system of [15]. This system detects parts of documents that may correspond to symbols, without a priori knowledge about the type of the

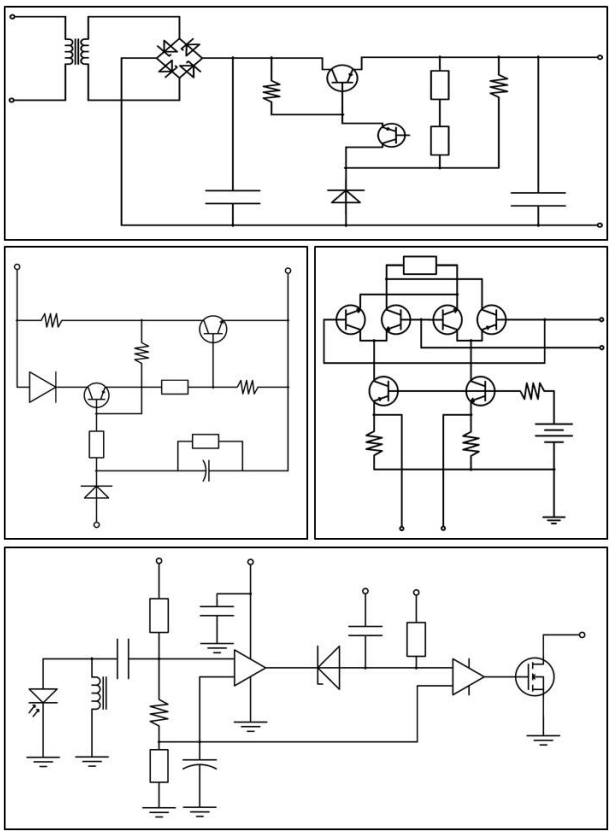


Fig. 28 Examples of built diagrams

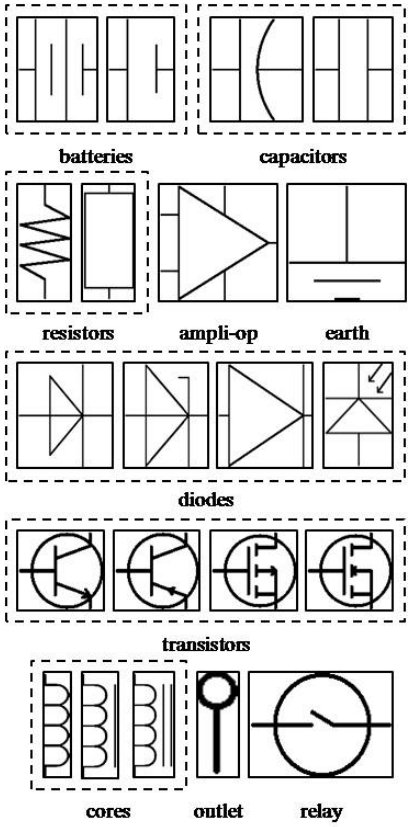


Fig. 29 Electrical models

document. It provides a set of ROIs[15] corresponding to potential symbols, without any information about their classes. It relies on a structural approach using a two-step process.

First, it extracts topological and geometric features from an image, and organizes them through an ARG[16] (Figure 30). The image is first vectorized into a set of quadrilateral primitives. These primitives become nodes in the ARG (labels 1, 2, 3, 4), and the connections between them, arcs. Nodes have, as attributes, relative lengths (normalized between 0 and 1) whereas arcs have the connection-types (L *junction*, T *junction*) and relative angles (normalized between 0° and 90°).

In the second step, the system looks for potential ROIs corresponding to symbols in the image. It detects parts of the graph that may correspond to symbols i.e. symbol seeds. Scores, corresponding to probabilities of being part of a symbol, are computed for all edges and nodes of the graph. They are based on features such as lengths of segments, perpendicular and parallel angular relations, degrees of nodes, etc. The symbol seeds are detected next during a score propagation process. This process seeks and analyses the different shortest paths and loops between nodes in the graph. The scores of all the nodes belonging to a detected path are homogenized

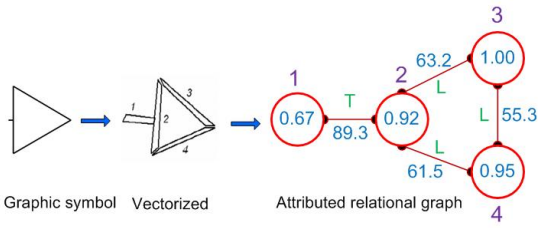


Fig. 30 Representation phase of [15]

(propagation of the maximum score to all the nodes in the path) until convergence, to obtain the seeds.

To evaluate this system we have constituted a dataset of synthetic documents using our collections. Table 4 gives details about it. It is composed of architectural floorplans and electrical diagrams. We have selected 100 drawings produced using 5 different background images (20 drawings are generated per background image) for each domain. Thus, the whole dataset contains 200 documents composed of 3861 symbols (2521 architectural and 1340 electrical). These symbols belong to 16 and 17 architectural and electrical models, respectively appearing in the selected backgrounds.

Our key objective here is to illustrate that our documents are suitable for performance evaluation. To do it, we propose to analyze the variability of the localization results, obtained by the system [15], on our dataset. We compute for each test image a symbol detection rate.

[15] Regions Of Interest
[16] Attributed Relational Graph

		Drawing level		Symbol level	
floorplans	Setting	backgrounds	5	models	16
	Dataset	images	100	symbols	2521
diagrams	Setting	backgrounds	5	models	17
	Dataset	images	100	symbols	1340

Table 4 Dataset used for experiments

single	an object in the results matches with a single object in the groundtruth
misses	an object in the groundtruth doesn't match with any object in the results
false alarm	an object in the results doesn't match with any object in the groundtruth
multiple	an object in the results matches with multiple objects in the groundtruth (merge case) or an object in the groundtruth matches with multiple objects in the results (split case)

Table 5 Matching cases between groundtruth and results

This rate is obtained by comparing the bounding boxes of the localization results and the groundtruth (see Annex B5 for details about the overlapping test). We exploit the overlapping relations to identify matching cases as detailed in Table 5. The detection rate of a given document corresponds to $d = \frac{s}{n}$, with s the number of single localization, and n the number of symbols in groundtruth.

Figure 31 presents plots of results we have obtained on architectural floorplans and electrical diagrams. In these plots, the detection rates are grouped per drawings produced from the same background image. Each curve gives the rates for a set of document instances (i.e. documents generated from a same background). Each set is composed of 20 document instances, and each plot gives results for 5 sets for an overall number of 100 documents. To draw the curves, we have sorted the detection rates per set from highest to lowest values.

These curves illustrate the variability of synthetic documents produced using our system. Variations in the symbol layer impact in a significative way the results of the system, despite the use of a same background image to produce the drawings. In order to quantify this variability, we present in Table 6 a statistical analysis. For each drawing set, we have computed the mean detection rates μ_b and their corresponding standard deviations σ_b. $\overline{\sigma_b}$ is the mean standard deviation of the drawing sets, whereas σ_w is obtained from the whole dataset (comparing documents generated from different backgrounds). In these experiments we obtain $\overline{\sigma_b} \simeq \frac{1}{2}\sigma_w$. These results show that our method can produce, for a given background, drawings with different features. Depending on the number and type of symbols and constraints,

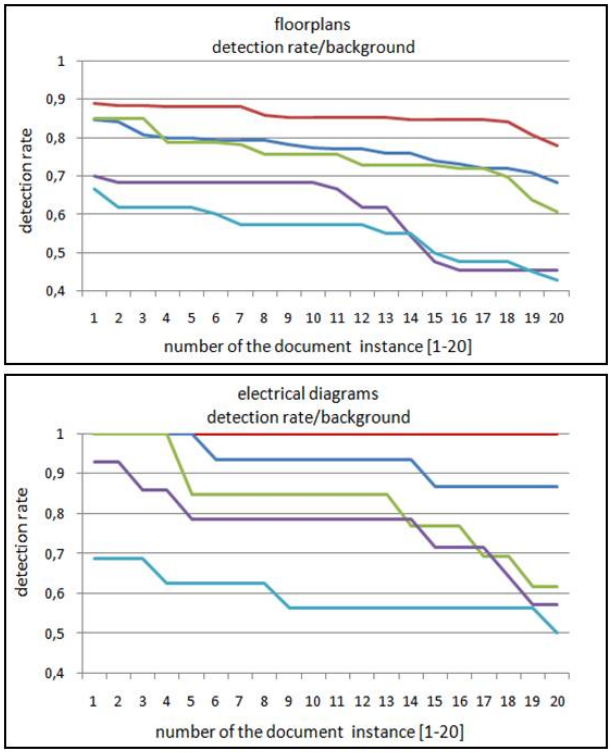


Fig. 31 Localization results (plots)

floorplans

	μ_b	σ_b
b1	0,8552	0,0273
b2	0,7691	0,0428
b3	0,7503	0,0629
b4	0,6018	0,1025
b5	0,5539	0,0654

$\sigma_w = 0{,}1247$ $\overline{\sigma_b} = 0{,}0602$

diagrams

	μ_b	σ_b
b1	1,0000	0,0000
b2	0,9300	0,0506
b3	0,8269	0,1167
b4	0,7679	0,0952
b5	0,5938	0,0517

$\sigma_w = 0{,}1568$ $\overline{\sigma_b} = 0{,}0628$

$$\overline{\mu_b} = \sum_{i=1}^{n} \mu_{bi} \qquad \sigma_w = \frac{\sum_{i=1}^{n} \left(\mu_{bi} - \overline{\mu_b}\right)^2}{n} \qquad \overline{\sigma_b} = \sum_{i=1}^{n} \sigma_{bi}$$

Table 6 Localization results (tables)

these features can significantly affect the performance of symbol localization and thus, be used for performance evaluation.

6 Conclusion and Perspectives

In this paper, we have presented a system for the generation of synthetic documents for the performance evaluation of symbol recognition & spotting systems. Our key contribution is the building of whole documents (drawings, maps, diagrams, etc.) and, our underlying aim, to

make these documents more realistic. To do it, we have exploited the layer property of graphical documents in order to position symbol sets in different ways using the same backgrounds. This way, we obtain a large amount of documents that look realistic by simply providing, a small number of constraints, and a few predefined backgrounds. The groundtruth (the locations and the labels of symbols) becomes automatically available along with the produced documents. We have employed this system to produce documents of architectural and electronic domains which proves the flexibility of our approach. In addition, using these documents we have done performance evaluation experiments of a symbol localization system. They show that the different parameters used in our system result in generated documents with different features, that are reflected in variation of the localization results of the system.

As a continuation of this work, our challenge is the comparison of synthetic documents with real ones in terms of performance evaluation. The goal will be to compare localization results, of one (or several) system(s), on real and on synthetic documents. The generation parameters (number and type of symbols, constraints and noise) used in our system, should establish some kind of relation and comparison to the same parameters in real documents. However, this is not an straightforward task due to the lack of groundtruthed datasets of real documents and characterization methods. To the best of our knowledge, it is not possible to achieve such a comparison today, because no groundtruthed dataset of real scanned documents exists. In addition, performance characterization metrics must be reformulated to take specificities of whole documents into account. With whole documents, characterization becomes harder because it has to be done between symbol sets. These symbol sets can have different sizes, and gaps can also appear concerning the locations of symbols. Different matching cases then exist, and the characterization methods must be able to detect them properly. Some recent research work on these problems can be found in [34,35]. Additional methods should be proposed by the graphics recognition community and compared, in order to identify the best-suited one for performance characterization.

7 Acknowledgements

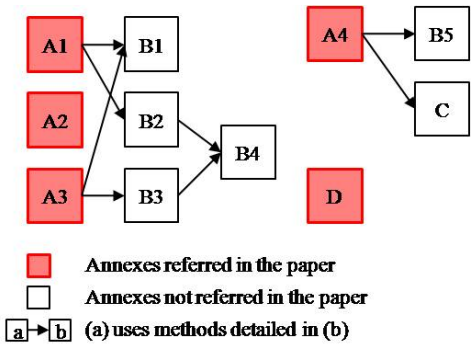
This work was partially supported by the Spanish project TIN2006-15694-C02-02, the fellowship 2006 BP-B1 00046 and by the Spanish research programme Consolider Ingenio 2010: MIPRCV (CSD2007-00018). The authors wish to thank Jean-Yves Ramel (LI laboratory, Tours, France) for his collaborations and help about this work.

References

1. E. Greengrass, Information retrieval: A survey, Tech. Rep. TR-R52-008-001, Center for Architectures for Data-Driven Information Processing (CADIP), University of Maryland, US (2000).
2. N. Thacker, A. Clark, J. Barron, j.R. Beveridge, P. Courtney, W. Crum, V. Ramesh, C. Clark, Performance characterisation in computer vision: A guide to best practices, Computer Vision and Image Understanding (CVIU) 109 (2008) 305–334.
3. H. Muller, W. Muller, D. Squire, S. Marchand-Maillet, T. Pun, Performance evaluation in content-based image retrieval: Overview and proposals, Pattern Recognition Letters (PRL) 22 (5) (2001) 593–601.
4. R. Haralick, Performance evaluation of document image algorithms, in: Workshop on Graphics Recognition (GREC), Vol. 1941 of Lecture Notes in Computer Science (LNCS), 2000, pp. 315–323.
5. A. Chhabra, Graphic symbol recognition: An overview, in: Workshop on Graphics Recognition (GREC), Vol. 1389 of Lecture Notes in Computer Science (LNCS), 1998, pp. 68–79.
6. L. Cordella, M. Vento, Symbol and shape recognition, in: Workshop on Graphics Recognition (GREC), Vol. 1941 of Lecture Notes In Computer Science (LNCS), 1999, pp. 167–182.
7. J. Lladós, E. Valveny, G. Sánchez, E. Martí, Symbol recognition : Current advances and perspectives, in: Workshop on Graphics Recognition (GREC), Vol. 2390 of Lecture Notes in Computer Science (LNCS), 2002, pp. 104–127.
8. K. Tombre, S. Tabbone, P. Dosch, Musings on symbol recognition, in: Workshop on Graphics Recognition (GREC), Vol. 3926 of Lecture Notes in Computer Science (LNCS), 2005, pp. 23–34.
9. S. Yoon, G. Kim, Y. Choi, Y. Lee, New paradigm for segmentation and recognition, in: Workshop on Graphics Recognition (GREC), 2001, pp. 216–225.
10. K. Tombre, B. Lamiroy, Graphics recognition - from re-engineering to retrieval, in: International Conference on Document Analysis and Recognition (ICDAR), 2003, pp. 148–155.
11. P. Dosch, J. Lladós, Vectorial signatures for symbol discrimination, in: Workshop on Graphics Recognition (GREC), Vol. 3088 of Lecture Notes in Computer Science (LNCS), 2004, pp. 154–165.
12. S. Tabbone, L. Wendling, D. Zuwala, A hybrid approach to detect graphical symbols in documents, in: Workshop on Document Analysis Systems (DAS), Vol. 3163 of Lecture Notes in Computer Science (LNCS), 2004, pp. 342–353.
13. D. Zuwala, S. Tabbone, A method for symbol spotting in graphical documents, in: Workshop on Document Analysis Systems (DAS), Vol. 3872 of Lecture Notes in Computer Science (LNCS), 2006, pp. 518–528.
14. H. Locteau, S. Adam, E. Trupin, J. Labiche, P. Heroux, Symbol spotting using full visibility graph representation, in: Workshop on Graphics Recognition (GREC), 2007, pp. 49–50.
15. R. Qureshi, J. Ramel, D. Barret, H. Cardot, Symbol spotting in graphical documents using graph representations, in: Workshop on Graphics Recognition (GREC),

Vol. 5046 of Lecture Notes in Computer Science (LNCS), 2008, pp. 91–103.
16. M. Rusiñol, J. Lladós, A region-based hashing approach for symbol spotting in technical documents, in: Workshop on Graphics Recognition (GREC), Vol. 5046 of Lecture Notes in Computer Science (LNCS), 2008.
17. E. Valveny, al., A general framework for the evaluation of symbol recognition methods, International Journal on Document Analysis and Recognition (IJDAR) 1 (9) (2007) 59–74.
18. S. Aksoy, al., Algorithm performance contest, in: International Conference on Pattern Recognition (ICPR), Vol. 4, 2000, pp. 870–876.
19. E. Valveny, P. Dosch, Symbol recognition contest: A synthesis, in: Workshop on Graphics Recognition (GREC), Vol. 3088 of Lecture Notes in Computer Science (LNCS), 2004, pp. 368–386.
20. P. Dosch, E. Valveny, Report on the second symbol recognition contest, in: Workshop on Graphics Recognition (GREC), Vol. 3926 of Lecture Notes in Computer Science (LNCS), 2006, pp. 381–397.
21. E. Valveny, P. Dosch, A. Fornes, S. Escalera, Report on the third contest on symbol recognition, in: Workshop on Graphics Recognition (GREC), Vol. 5046 of Lecture Notes in Computer Science (LNCS), 2008, pp. 321–328.
22. D. Lopresti, G. Nagy, Issues in ground-truthing graphic documents, in: Workshop on Graphics Recognition (GREC), Vol. 2390 of Lecture Notes in Computer Science (LNCS), 2002, pp. 46–66.
23. L. Yan, L. Wenyin, Interactive recognizing graphic objects in engineering drawings, in: Workshop on Graphics Recognition (GREC), Vol. 3088 of Lecture Notes in Computer Science (LNCS), 2004, pp. 126–137.
24. A. Chhabra, I. Phillips, The second international graphics recognition contest - raster to vector conversion : A report, in: Workshop on Graphics Recognition (GREC), Vol. 1389 of Lecture Notes in Computer Science (LNCS), 1998, pp. 390–410.
25. J. Zhai, L. Wenyin, D. Dori, Q. Li, A line drawings degradation model for performance characterization, in: International Conference on Document Analysis and Recognition (ICDAR), 2003, pp. 1020–1024.
26. B. Yanikoglu, L. Vincent, Pink panther: a complete environment for ground-truthing and benchmarking document page segmentation, Pattern Recognition (PR) 31 (9) (1998) 1191–1204.
27. C. Lee, T. Kanungo, The architecture of trueviz: A groundtruth/metadata editing and visualizing toolkit, Pattern Recognition (PR) 36 (3) (2003) 811–825.
28. A. Antonacopoulos, D. Karatzas, D. Bridson, Ground truth for layout analysis performance evaluation, in: Workshop on Document Analysis Systems (DAS), Vol. 3872 of Lecture Notes in Computer Science (LNCS), 2006, pp. 302–311.
29. D. Kim, , T. Kanungo, Attributed point matching for automatic groundtruth generation, International Journal on Document Analysis and Recognition (IJDAR) 5 (1) (2002) 47–66.
30. G. Ford, G. Thoma, Ground truth data for document image analysis, in: Symposium on Document Image Understanding and Technology (SDIUT), 2003, pp. 199–205.
31. L. Yang, W. Huang, C. Tan, Semi-automatic ground truth generation for chart image recognition, in: Workshop on Document Analysis Systems (DAS), Vol. 3872 of Lecture Notes in Computer Science (LNCS), 2006, pp. 324–335.
32. I. Phillips, J. Ha, R. Haralick, D. Dori., The implementation methodology for the cd-rom english document database, in: International Conference on Document Analysis and Recognition (ICDAR), 1993, pp. 484–487.
33. T. Kanungo, R. Haralick, h.S. Baird, W. Stuezle, D. M. and, A statistical, nonparametric methodology for document degradation model validation, Pattern Analysis and Machine Intelligence (PAMI) 22 (11) (2000) 1209–1223.
34. M. Delalandre, J. Ramel, E. Valveny, M. Luqman, A performance characterization algorithm for symbol localization, in: Workshop on Graphics Recognition (GREC), Vol. 8, 2009, pp. 3–11.
35. M. Rusiñol, J. Lladós, A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices, International Journal on Document Analysis and Recognition (IJDAR) 12 (2) (2009) 83–96.

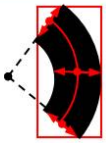
Organization of the Annexes



Annex A: Positioning Methods

A1: Computation of bounding boxes

bounding box is a min-max of straight-line corners, corners are offsets of extremities in the line direction at (+/-) π/2

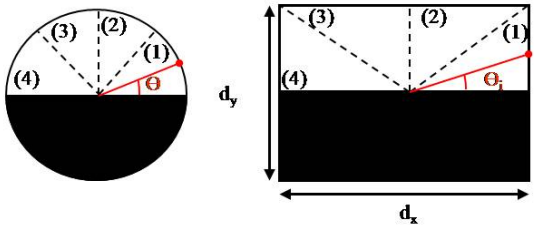


same as straight-line, in addition we look for the cardinal extremities of the arc

See Annexes B1 and B2 for the *point offset* and *direction inclusion test within an arc.*

A2: Computation of control points

how to get the direction θ_i from θ ?



(1) $\theta_i = a\tan\left(\tan(\theta) \times d_y / d_x\right)$

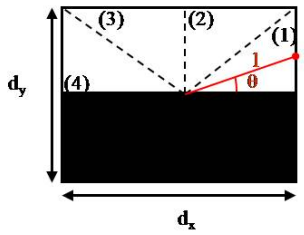
(2) $\theta_i = \frac{\pi}{2} - a\tan\left(\tan\left(\frac{\pi}{2} - \theta\right) \times d_x / d_y\right)$

(3) $\theta_i = \frac{\pi}{2} + a\tan\left(\tan\left(\theta - \frac{\pi}{2}\right) \times d_x / d_y\right)$

(4) $\theta_i = \pi - a\tan\left(\tan(\pi - \theta) \times d_y / d_x\right)$

how to get the length L_i from L ?

$\theta = \pi - a\tan\left(\frac{d_y}{d_x}\right)$ $\theta = \frac{\pi}{2}$ $\theta = a\tan\left(\frac{d_y}{d_x}\right)$



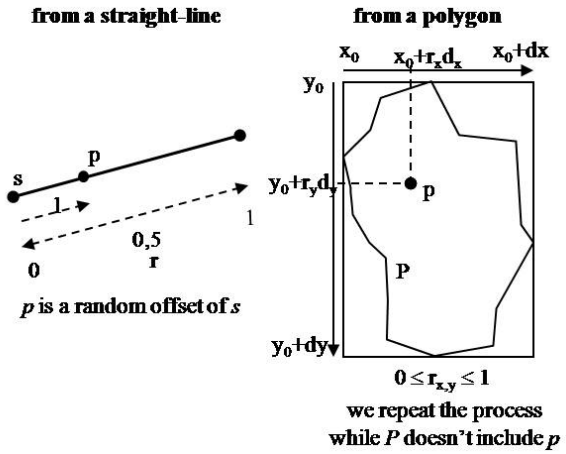
(1) $l = \frac{d_x}{2 \times \cos(\theta)}$

(2) $l = \frac{d_y}{2 \times \cos\left(\theta - \frac{\pi}{2}\right)}$

(3) $l = \frac{d_y}{2 \times \cos\left(\frac{\pi}{2} - \theta\right)}$

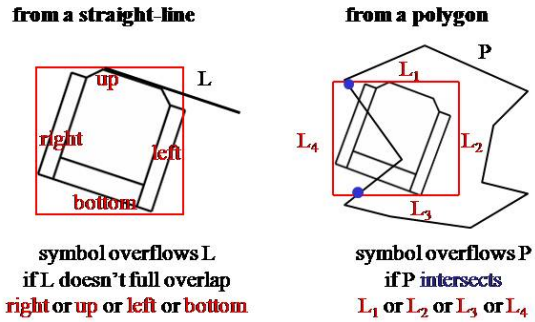
(4) $l = \frac{d_x}{2 \times \cos(\pi - \theta)}$

A3: Generation of random points



See Annexes B1 and B3 for the *point offset* and *point inclusion test within a polygon.*

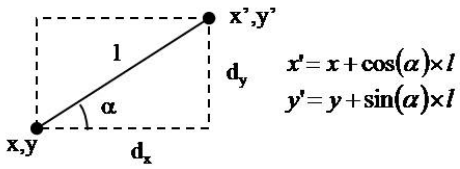
A4: Overflow tests



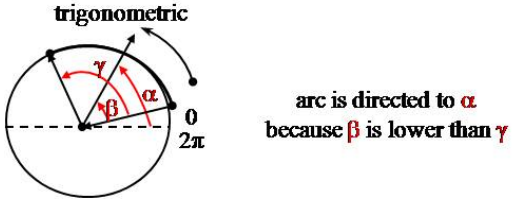
See Annexes B5 and C for the *overlapping tests* and *line intersection methods.*

Annex B: Computational Geometry Methods

B1: Point offset

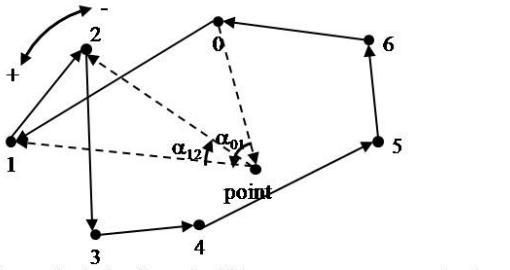


B2: Direction inclusion test within an arc



See Annex B4 for the *clockwise angle computation between straight-lines.*

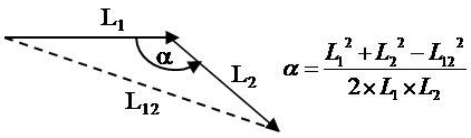
B3: Point inclusion test within a polygon



See Annex B4 for the *clockwise angle computation between straight-lines.*

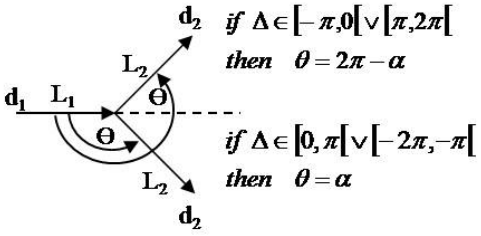
B4: Clockwise angle computation between straight-lines

L_1 and L_2 are two lines and α their angle gap

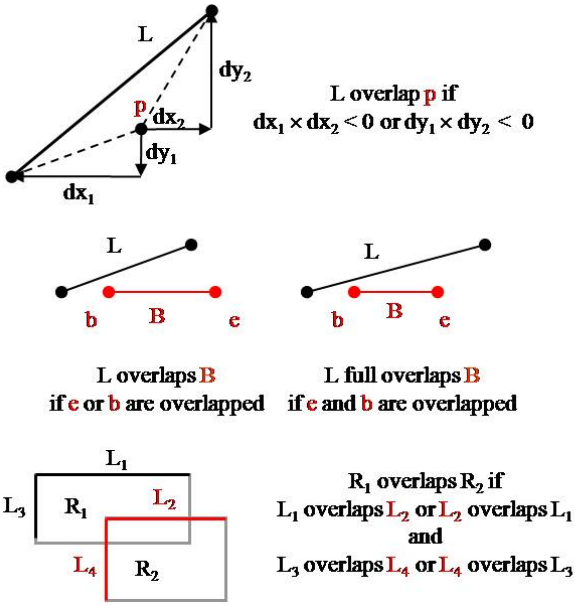


d_1 and d_2 are the two lines' directions in [0- 2π]

$$\Delta = d_1 - d_2$$

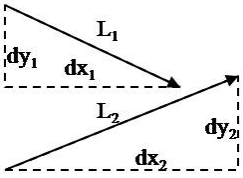


B5: Overlapping tests



Annex C: Line Intersection Methods

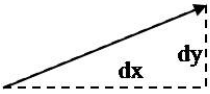
test of line intersection



L_1 and L_2 intersect if

$$dx_1 \times dy_2 - dx_2 \times dy_1 \neq 0$$

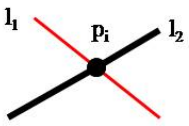
case of line intersection



oblique	$dx \neq 0$	$dy \neq 0$
vertical	$dx = 0$	
horizontal	$dy = 0$	

computation of line intersection

oblique-oblique



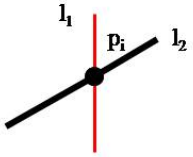
l_1 $y = m_1 \times x + p_1$

l_2 $y = m_2 \times x + p_2$

p_i $x_i = \dfrac{p_2 - p_1}{m_1 - m_2}$

$y_i = \dfrac{m_2 \times p_1 - p_2 \times m_1}{m_2 - m_1}$

oblique-vertical



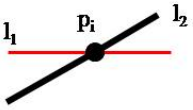
l_1 $x = p_1$

l_2 $y = m_2 \times x + p_2$

p_i $x_i = p_1$

$y_i = m_2 \times p_1 + p_2$

oblique-horizontal



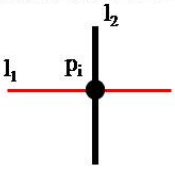
l_1 $y = p_1$

l_2 $y = m_2 \times x + p_2$

p_i $x_i = \dfrac{p_1 - p_2}{m_2}$

$y_i = p_1$

vertical-horizontal



l_1 $y = p_1$

l_2 $x = p_2$

p_i $x_i = p_2$

$y_i = p_1$

Annex D: Groundtruth Format

```
−<symbols>
    <symbol label="door1" x0="1289.66" y0="892.94" x1="1565.07"
    y1="1166.71" scale="4.58" morph="5.28" direction="0.0"/>
    <symbol label="door1" x0="1276.01" y0="1885.14" x1="1551.43"
    y1="2158.91" scale="4.58" morph="5.28" direction="180.0"/>
    <symbol label="window2" x0="4399.07" y0="77.82" x1="4622.73"
    y1="120.92" scale="3.62" morph="5.28" direction="0.0"/>
    <symbol label="window1" x0="2438.16" y0="67.73" x1="2661.81"
    y1="128.33" scale="3.63" morph="5.28" direction="0.0"/>
    <symbol label="door1" x0="985.83" y0="1236.97" x1="1259.59"
    y1="1512.39" scale="4.58" morph="5.28" direction="270.0"/>
    <symbol label="door1" x0="985.31" y0="1567.07" x1="1259.07"
    y1="1842.49" scale="4.58" morph="5.28" direction="270.0"/>
</symbols>
```