Building Synthetic Graphical Documents for Performance Evaluation

Mathieu Delalandre[1], Tony Pridmore[2], Ernest Valveny[1],
Hervé Locteau[3], and Eric Trupin[3]

[1] CVC, Barcelona, Spain
{mathieu;ernest}@cvc.uab.es
[2] SCSIT, Nottingham, England
tony.pridmore@nottingham.ac.uk
[3] LITIS, Rouen, France
{herve.locteau;eric.trupin}@univ-rouen.fr

Abstract. In this paper we present a system that allows its use to build synthetic graphical documents for the performance evaluation of symbol recognition systems. The key contribution of this work is the building of whole documents like drawings or maps. We exploit the layer property of graphical documents by putting symbol sets in different ways from a same background using positioning constraints. Experiments are presented to build two kinds of test document databases: bags of symbol and architectural drawings.

1 Introduction

In recent years there has been a noticeable shift of attention within the graphics recognition community to the topic of performance evaluation. Performance evaluation is divided into two main topics: ground-truthing and performance characterization. The first is concerned with the production of test document databases and their corresponding ground-truth [1], while the second deals with the matching of system results to that ground-truth [2]. In this paper we are more interested in ground-truthing. Two main approaches exist in the literature: based on real-life or on synthetic documents.

The approach based on real-life documents is the most common [1]. Representative documents are obtained from paper archives and digital libraries and ground-truth is created and edited manually using suitable graphic user interfaces. This kind of ground-truthing results in realistic and unbiased data but raises different problems: how to define the ground-truth, how to deal with the errors introduced by users, the delay and cost of acquisition, the copyright of documents, and the effort required to constitute large databases. In many cases these problems render the approach impractical.

A complementary approach, which avoids these difficulties, is to create and use synthetic documents. Here, the test documents are built by an automatic system which combines pre-defined models of document components in a pseudo-random way. Test documents and ground-truth can therefore be produced simultaneously. In addition, a large number of documents can be generated easily and with limited user involvement. This topic is emerging and only the systems of [3] [2] [4] [5] [6] exist in the literature. Figure 1 gives some examples of document produced by these systems.

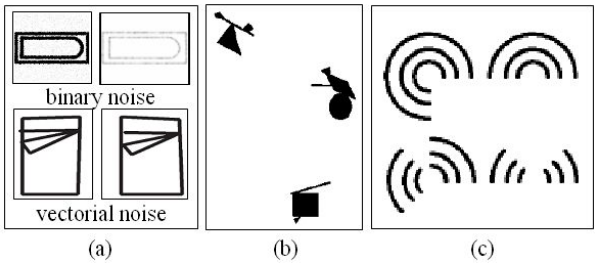

Fig. 1. Examples of synthetic document
(a) segmented symbol (b) random symbol set (c) arc set

The systems proposed by [5] and [6] support the generation of degraded images of segmented symbols as shown in the Figure 1 (a). In these systems, the symbol models are described in a vector graphics format. The vector graphics files are then converted into images. Two kinds of noise are added: binary [5] [6] and vectorial [6]. The system described in [4] employs a complementary approach to build documents composed of multiple unconnected symbols. The Figure 1 (b) gives an example of built document. Each symbol is composed of a set of primitive (circles, lines, squares, ...) randomly selected and mildly overlapped. They are next put on the image at a random location and without overlapping with the bounding boxes of other symbols. Finally, noise is added to the generated images using a binary distortion method. The systems of [3] [2] are similar to previous ones but use dynamic models. These dynamic models are described in a mathematical formalism which allows the composition of various graphical shapes. In [3] two models are proposed to generate the land parcels and houses that appear in cadastral maps. In [2] another model is defined and used to generate images composed of several arcs of different lengths and radius. The Figure 1 (c) illustrates the kind of image produced using this model. The arc images are then degraded by using a binary degradation algorithm.

All these systems are interesting, but in order to do a complete evaluation of graphics recognition systems we need whole documents. Indeed, real-life documents (like the engineering and architectural drawings, or the electrical diagrams) are composed of multiple objects constrained by spatial relations (connectivity, adjacency, neighbourhood, ...). The design of a suitable process to build such documents is a challenging task. Indeed, realistic documents can't be produced without human know how into the process. In our work we have considered a shortcut way to solve this problem. Our key idea is based on the property of graphical documents that are composed of two layers: a linear and a symbolic one. We use then this property to build several document instances: *ie* symbol sets positioned in different ways from a same background as shown in the Figure 2. Like this, the building process of whole document is made easier and can be considered as a positioning problem of symbols on a document background.

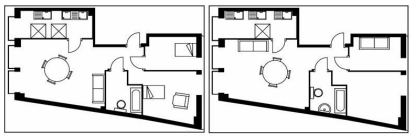

Fig. 2. Two document instances

The main architecture of our system is presented in the Figure 3. This uses as entry data a background image, a database of symbol models and a file containing the positioning constraints. These positioning constraints are edited by a user from the used background image and the models of symbol to associate. Based on these entries two main processes are exploited by our system to produce the document instances: a symbol factory and positioning. In what follows we present each of them in the sections (2) and (3). In section (4) we present our building manager supervising these two processes. Section (5) describes initial experiments and results we produce. Finally, in section (6) we conclude and give our perspectives.

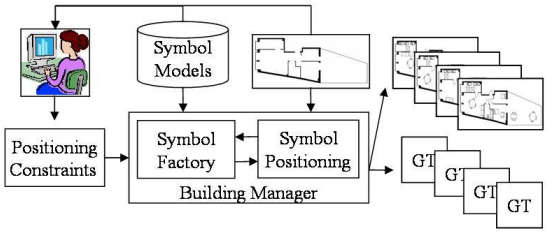

Fig. 3. Our system

2 Symbol Factory

Following the systems proposed by [5] and [6] we use geometrical primitives (straight lines, arcs and circles) and their associated thickness attributes to describe the symbol models. Each model is then stored in an individual file kept inside a database. The user accesses the contents of the database by defining in the file of positioning constraints the models he/she wants to use. Obviously, in order to produce different document instances, these models are selected at random. The system assigns previously a selection probability for each of them. In our system we have computed these probabilities by taking into account the existing links between the symbols and our positioning constraints[4]. These links are defined in the constraint file edited by the user as illustrated in the Figure 3. Each symbol referred in this file is associated to one or several constraint(s) in order to coerce its positioning on the document.

[4] These constraints are of three types, we will present each of them in the next section 3.

Figure 4 details our selection process. We compute first for each constraint a weight w_c from its number of associated symbols n_s. The obtained weights are then used to compute the symbol ones w_s by summing their associated w_c. Finally, these w_s are normalized by the total number of constraints n_c to obtain the inclusion probabilities p_s of each symbol . This approach increases the inclusion probabilities of symbols linked to several constraints (like s_3) or those linked to weak associated constraints (like s_6). In this way, the selection will respect a distribution of symbols among the constraints.

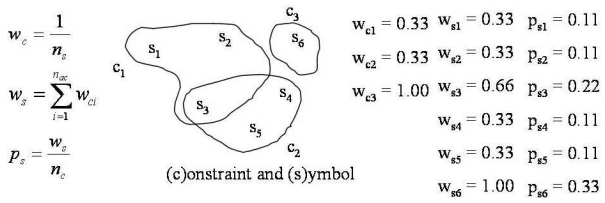

Fig. 4. Symbol selection

Once selected we load the symbols from their model files, scale them to adapt them to the background's size, and compute their bounding boxes. Indeed, the bounding boxes are a common way to handle graphical objects inside a document analysis system. In ours we use them during the positioning process. Computing a bounding box from a set of straight lines is an easy thing to do. However, in our system in addition to straight lines we also use arcs and circles. Moreover, our primitives are given thickness attributes. In order to take this into account we use two methods. The first presented in the Figure 5 (a) is a classical projection method to move a point according to a given length and direction. This method allows us to find the corners of thick straight lines, and then to compute their bounding boxes. When processing arcs we also search for the corners but also the cardinal points (west, east, north and south). Indeed, these cardinal points are necessary to compute the full bounding box of an arc. To compute these points we use a direction test detailed in the Figure 5 (b). This detects if right angles $\{0, \frac{\pi}{2}, \pi, \frac{3\times\pi}{2}\}$ belong to the arc. It raises on a method to compute the trigonometric angles between vectors detailed on the Figure 5 (c).

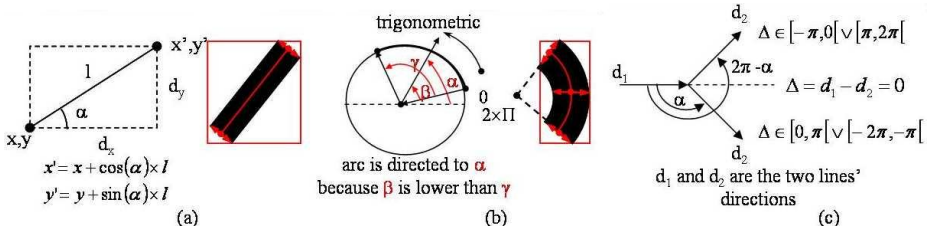

Fig. 5. Computation of bounding box
(a) point projection (b) direction test (c) computation of trigonometric angle

3 Symbol Positioning

Following the factory process we position the built symbols on the background to produce the document instances. In order to fill correctly the background, constraints must be used to coerce the positioning of symbols. In our approach we have considered three types of constraint: fixed, sliding and zone. We present each of them in what follows.

Our first positioning constraint is the fixed one. Its mechanism is presented in the Figure 6. It raises on the matching between two points: a control point on the built symbol and an anchor one defined on the background. The built symbol is then positioned in such way on the background that the control point will match with the anchor one. However, in order to extend the positioning possibilities of this constraint we also permit on a symbol to define a particular control point and to apply a rotation transformation. The symbol is then positioned in three steps: it is rotated (using a parameter that can be null, a fixed value or a range), its control point is computed, it is at last fixed on the anchor point using the control one.

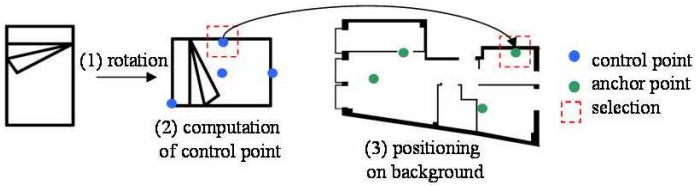

Fig. 6. Fixed constraint

The key step of this process is the computation of the control point. This point is defined in each constraint by unit polar coordinates (ρ, θ) from the center of the bounding box. These unit polar coordinates are used to compute the values of length and direction (l, α) used to project the center of bounding box to obtain a control point as explained in the Figure 7 (a). The α value is equal to $\theta \times 2\pi$. The l one is computed in different ways (1,2,3 and 4) according to the sizes of bounding box's sides and weighted at last by ρ. The Figure 7 (b) gives some examples of positioning around an anchor point using $\rho = 1$ and $\theta = \{0, \frac{3}{20}, \frac{6}{20}, \frac{9}{20}, \frac{3}{20}, \frac{15}{20}, \frac{18}{20}\}$. The figures (c) and (d) gives examples using previous rotations of symbol and with control points defined by $(\rho = 1.0, \theta = 0.25)$ and $(\rho = 1.0, \theta = 0.75)$.

Our second positioning constraint is the sliding one. This constraint extends the fixed one by using an anchor point taken at random along a line. The symbol is positioned in five steps: it is rotated in the same way as in a fixed constraint, its control point is computed, the symbol and its control point are rotated in the line's direction, a random anchor point is taken at random along the line, the symbol is at last fixed on the anchor point. The computation of the random anchor point along the line is simply done by generating a random number r as shown Figure 8 (a). This number r is used to compute the length l between the line's begin b and the random point p. The beginning point b is next projected to obtain p using the length l and the line's direction. The Figure 8 (b) gives an example of symbol slided along a line.

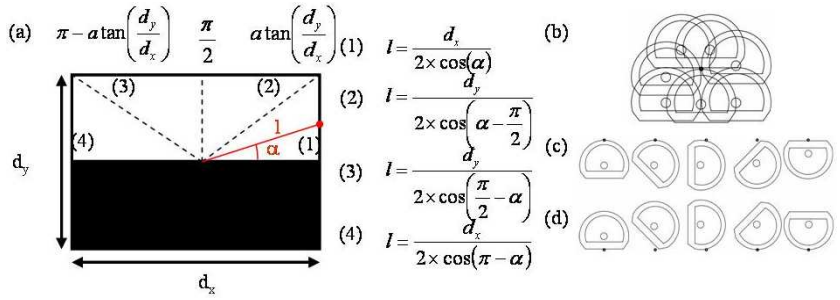

Fig. 7. Control point
(a) computation (b) (c) (d) examples of result

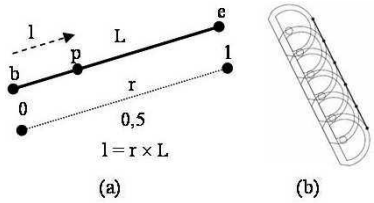

Fig. 8. Sliding constraint
(a) computation of random point (b) example of result

Our last positioning constraint deals with a specific zone inside a document. Its purpose is to constraint the positioning of symbols to this zone. To do this we define our zones with polygons. From a polygon the positioning is done in two steps: a random anchor point (including the polygon boundary) is taken at random, the symbol is next fixed on this anchor point in the same way as in a fixed constraint. The specificity of this process is then the generation of a random anchor point including the polygon boundary. The method that we use is decomposed in three steps. It starts by computing the bounding box of a polygon using min max methods from the x,y values of its points. We generate next a random point from this bounding box with two random numbers r_x and r_y as explained in the Figure 9 (a). Obviously it can appear that this point can be outside the polygon. To check it we use an inclusion test presented in the Figure 9 (b). This method sums the trigonometric angles (with the method described in the Figure 5 (c)) of successive vectors joining the random point and the polygon ones. A $2 \times \pi$ value corresponds then to an inclusion case. We repeat the generation process from the bounding box while the inclusion test is negative. Figure 9 (c) presents an example of inclusion result from random generated polygon and point set.

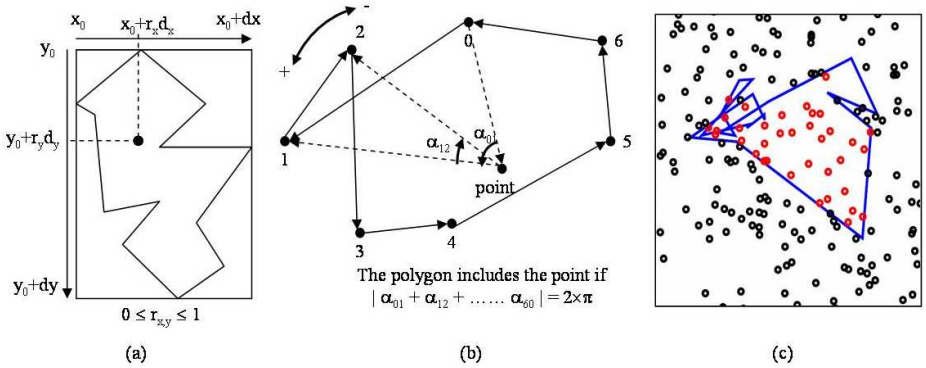

Fig. 9. Zone constraint
(a) random point from a bounding box (b) inclusion test (c) example of result

4 Building Manager

In the proposed system the factory and positioning processes are managed by an explicit document building process. This starts with empty documents and fills them, in a pseudo-random way, with built symbols. The user defines, in the constraint file, the number of symbols that he wants to include in the document. The process will stop when this number is reached.

However, it may be that a positioning can fail. These failures appear for example when a symbol is positioned to overlap an existing one, when parts of a symbol (other than the control point) overflow a constraint area, ... The system must be able to identify these failures to cancel the positioning. Moreover, users might define a number of symbols per document that will be hard to satisfy without strapping the defined constraints. The system must then detect this case in order to avoid an infinite building process. To solve these problems in our system we use four tests, three to check the positioning of symbols and one to stop the whole building process.

Our first positioning checking concerns the management of the free space of document. Indeed, during the building process several symbols can share a same place. In order to prevent such a case we test the overlapping between the bounding boxes of symbols. This test is computed in three steps as explained in Figure 10: first between a line and a point (a), then between two lines (b) and at last between the two bounding boxes (c). We test then the overlapping between a symbol to position with all the ones already positioned on the document. Any positive case produces a building failure.

Our second positioning checking deals with the sliding constraint as shown in the Figure 11. The positioning process of this constraint could produce overflows of symbols around the line's borders (a). In order to limit the positioning to the area of constraint's we have defined an overflow test. This test is based on the covering between two lines (b). This one is just a logical adaptation of the overlapping test previously presented in the Figure 10. A symbol can be then considered as overflowing if any of borders $\{right, up, left, bottom\}$ of its bounding box is not covered by the constraint's line L (c). A positive case produces then a building failure.

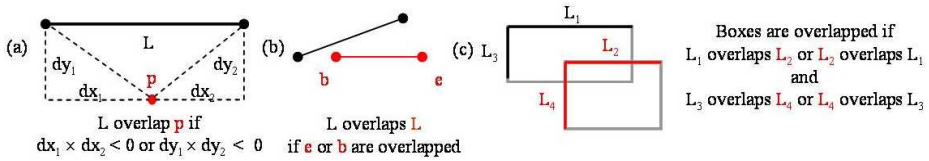

Fig. 10. Overlapping test
(a) line-point (b) line-line (c) box-box

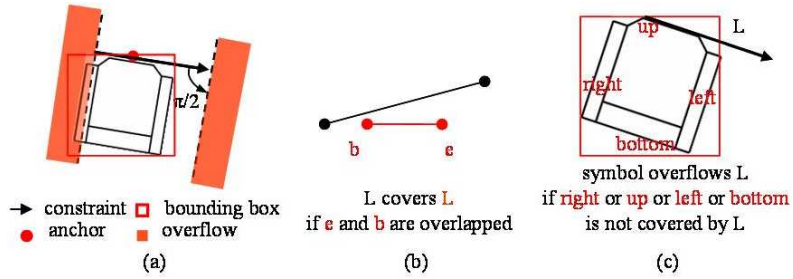

Fig. 11. Sliding checking
(a) symbol overflow (b) covering test (c) overflow test

Our last positioning checking is related to the zone constraint. Indeed, in the same way as in the sliding one, overflows of symbols can appear. The next Figure 12 (a) gives an example of this case. It corresponds to a random anchor point generated too near of borders of polygon. In order to detect such a case we exploit the bounding box's corners of the symbol as explained in the Figure Figure 12 (b). We test then if these corners are included in the polygon using our inclusion test presented in the Figure 9 (b). Any false case will produce a building failure.

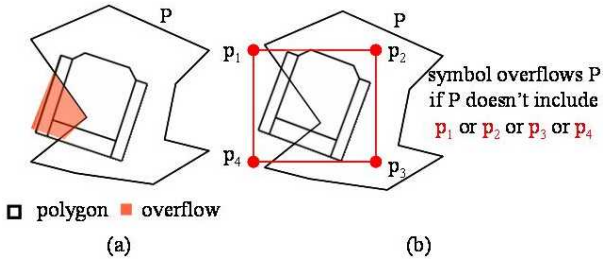

Fig. 12. Zone checking
(a) symbol overflow (b) including test of bounding box's corners

In a last test we control the progress of building process in order to stop it if necessary. Indeed, the system must then detect the number of building failure in order to avoid an infinite building process. To do this we use the number of symbol per document as stop criterion. We compare it with the number of building failures. When it becomes lower we stop the process.

5 Experiments and Results

In this section we present initial experiments and results of our system. The main objective of these experiments is to constitute databases of test document, with their corresponding ground-truth, for the third edition of the Symbol Recognition Contest held during the 2007 Workshop on Graphics Recognition (GREC'07)[5]. To do it we have used the symbol model library defined for the two previous editions of the Contest[6,7]. This is composed of 150 models of architectural and electrical symbols. Based on this library we have edited several constraint sets in order to build test document databases of different type. Obviously, the documents produced by our system are in a vector graphics form. For the Contest purposes these documents should therefore be converted into binary images; noise can be then added by the distortion methods used in the past editions of Contest[6,7] [6].

We have edited a first set of constraint in order to build "bag of symbol" documents. Figure 13 presents examples of these bags. In them the symbols are put at random on an empty background, without any connection, and using different rotation or scaling parameters. So these documents look similar to the ones generated by [4] (see Figure 1 (b)). However, they are composed of real-life symbols and not only of geometrical shapes. The key idea of this data set is to constitute an intermediate level of evaluation between the documents composed of a single segmented symbol (as proposed in the last editions of the Contest[6,7]) and whole documents (like drawings, maps or diagrams).

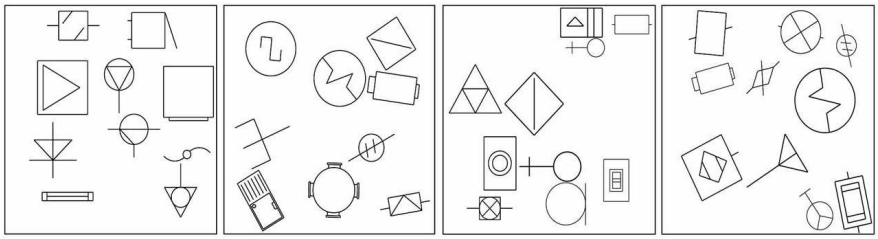

Fig. 13. Examples of bag of symbol
(a) none transformation (b) rotated (c) scaled (d) rotated & scaled

[5] http://www.buyans.com/grec2007/
[6] http://www.cvc.uab.es/grec2003/SymRecContest/
[7] http://symbcontestgrec05.loria.fr/

To generate these bags we have just defined in our setting a single squared zone constraint surrounding an empty background. In regard to the Contest organization[8] we have resized the original symbol models of past editions[6,7] from 512×512 to 128×128 pixels in order to build bags of reasonable dimension. Based on this initial size we have generated bags of 512×512 pixels composed of 10 symbols each. This corresponds to a mean symbol density of 0.625 ($\frac{128^2 \times 10}{512^2}$) which respects a good partitioning between the background and the foreground parts as shown in the Figure 13.

Using these size parameters we have generated 16 databases of 100 bags each. This corresponds to an overall number of 1600 bags composed of 16000 symbols. These 16 databases have been generated by respecting the protocol used during the previous editions of Contest[6,7]. First we have used different model numbers (25,50,100 and 150) in order to test the scalability of methods. The used model sets are the ones of the 2005 edition of Contest[7]. Next we have applied and combined different geometrical operations as illustrated in the Figures 13 (a), (b), (c) and (d). These transformation has been set as follow: from 0 to $2 \times \pi$ for the rotation with a gap of $\frac{2 \times \pi}{1000}$, and from 75 % to 125% for the scaling with a gap of 0.05 % ($\frac{50\%}{1000}$).

Our second set of constraints deals with the building of whole graphical documents using filled backgrounds. For that we have limited ourselves to the building of architectural drawings. The Figure 2 presented in the introduction section of this paper presents examples of the drawings we produce. We argue here that the positioning constraints presented in this paper are not domain dependant and could be re-used to build other kinds of document (electrical drawings, geographical maps, ...). However, the third edition of the Contest is a kickoff concerning the evaluation of whole documents. This constitutes an important gap for the systems and to limit the Contest to a single domain seems to be fair. We have chosen architectural drawings in recognition to their interesting properties concerning the connectivity and the orientation of symbols.

To generate these drawings we have retained the size parameter defined for our bags: 128×128 pixels per symbol. Obviously, the use of filled backgrounds makes the images produced bigger in regard to the one of bags. However, due to the Contest organization[8] we have fixed ourselves a limit of about 2048^2 pixels per image by considering only the backgrounds composed of a weak number of rooms (from 4 to 8). We have then selected 4 real-life drawings and constituted our backgrounds by cleaning their text and symbol parts with a vector graphics editor. From these backgrounds we have defined sets of constraint in order to generate databases of 30 images per background and 20 symbols per image. This corresponds to an overall number of 120 drawings composed of 2400 symbols. Obviously, to generate these drawings we have selected among the Contest's library[7] only the architectural models. This corresponds to an overall number of 16 models. The Figure 14 gives snapshots of these models with their corresponding labels. For all these models we have also defined resizing parameters, from 1.0 to 2.4, in order to respect the proportions between the symbols on the drawings. The resizing parameter of 1.0 corresponds then to symbols of 128×128 pixels and the uppers to the bigger symbols.

[8] It is done in real time during one session of the Workshop.

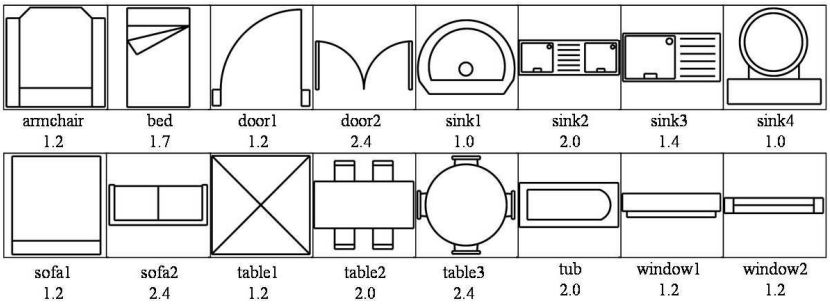

Fig. 14. Architectural symbols(labels & resizing parameters)

We have then used these models and their resizing parameters in our constraints. The number of defined constraints per background is about 20 and may be fixed, sliding or zone type. We have used the fixed constraint to put the door and window symbols on the drawings. The sliding constraint has allowed us to connect the symbols like the skins, the tubs or the beds along the walls. In each sliding constraint the symbols are put in the line direction and rotated using a gap of $\frac{\pi}{2}$ in order to respect the wall/symbol alignment. Finally, we have used the zone constraints to define the boundaries of rooms in order to position the other furnitures like the armchairs, the tables or the sofas. Inside, the symbols have been rotated from 0 to $2 \times \pi$ with a gap of $\frac{2 \times \pi}{1000}$.

6 Conclusion and Perspectives

In this paper we have presented a system that allows its use to build synthetic graphical documents for the performance evaluation of symbol recognition systems. Our main contribution is to extend the past works in this field to the building of whole documents (drawings, maps, diagrams, ...). To do it we have exploited the layer property of graphical documents in order to position symbol sets in different ways on the same background. Our approach raises on the use of constraint in order to coerce the positioning of symbols. The system that we propose is composed of three components: a symbol factory to select and to load the symbols, a symbol positioning to solve the constraints, and a building manager to supervise the whole process. Experiments done show how our system allows to produce large databases of document that look real.

Concerning the perspectives different works are planned in the short term. First a GUI[9] to edit the positioning constraints. It will speed up the editing process and help users to build their own databases. Next, based on this GUI we want to use our system to generate other kinds of document like the electrical drawings or the geographical maps. Finally, a performance characterization tool is now needed in order to compare the systems' results with the corresponding ground-truth. Finally, a more long-term perspective concerns the use of a grammar engine to check the built documents. The key idea will be to improve the symbol distribution with rules: e.g. a bed room is composed at least of one bed, and occasionally of one table or sofa.

[9] Graphics User Interface

7 Acknowledgements

The authors wish to thank Karim Zouba and Murielle Ramangaseheno (LITIS, Rouen University, France) for their contributions to this work. A part of this work was granted by the EPEIRES project[10] of the French Techno-Vision program 2005-2006. Finally, the authors wish also to thank Philippe Dosch (LORIA, Nancy University, France) for his packaging work of databases for the Symbol Recognition Contest'07.

References

1. 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.
2. L. Wenyin, D. Dori, Principles of constructing a performance evaluation protocol for graphics recognition algorithms, in: Performance Characterization and Evaluation of Computer Vision Algorithms, Springer Verlag Publisher, 1999, pp. 97–106.
3. D. Madej, A. Sokolowski, Towards automatic evaluation of drawing analysis performance: A statistical model of cadastral map, in: Inernational. Conference on Document Analysis and Recognition (ICDAR), 1993, pp. 890–893.
4. S. Aksoy, al, Algorithm performance contest, in: International Conference on Pattern Recognition (ICPR), Vol. 4, 2000, pp. 870–876.
5. 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.
6. 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.

[10] http://epeires.loria.fr