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 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 positioning 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

Performance evaluation of graphics recognition systems goes back to the middle of 90's [1]. At this period the graphics recognition community focussed its researches on the evaluation of vectorization processes for document re-engineering. In recent years there has been a noticeable shift of attention towards the evaluation of symbol recognition [2], especially through the four International Contests on Symbol Recognition at ICPR 2000[4], and GREC 2003, 2005 and 2007[5]. Performance evaluation is divided into two main topics: ground-truthing and performance characterization. The first one is concerned with the production of test document databases and their corresponding ground-truth [3], while the second deals with the matching of system results to that ground-truth [4]. In this paper we are more interested in ground-truthing, focussed on the symbol recognition. Three main approaches exist in the literature: based on paper, CAD[6] and synthetic documents.

The approach based on paper documents is the most common [3]. Representative documents are obtained from paper archives and digital libraries, and ground-truth is created and edited manually using suitable GUI[7]. This kind of ground-truthing results

[4] http://www.ee.washington.edu/research/isl/IAPR/ICPR00/
[5] http://epeires.loria.fr/
[6] Computer Aided Design
[7] Graphic User Interface

in realistic and unbiased data but raises different problems: how to define the ground-truth, how to deal with the errors introduced by the users, the delay and the cost of the groundtruth acquisition, etc. In many cases these problems render the approach impractical.

A complementary approach that overcomes these problems is to work directly from CAD documents. As these documents are already in a vector graphics form (SVG, CGM, DWG, etc.), it is possible to take advantage of a groundtruth already existing. The CAD documents are next converted into images for the evaluation. Such approach has been used in the past to evaluate the raster to vector conversion processes [5]. It avoids the groundtruthing step required with the scanned images but it still involves collecting the initial documents. This collecting process takes into account several issues [6]: the copyrights, the format registration (to valid, to convert, etc.), the database organization, finding the duplicates, editing the metadata, etc.

A final approach, which avoids all the 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 document can be generated easily and with limited user involvement. In the past some systems have been proposed to generate synthetic documents to evaluate the vectorization systems [7]. Concerning the symbol recognition this topic is emerging and only the systems described in [8] [9] [2] exist in the literature. Figure 1 gives some examples of document obtained with these systems.

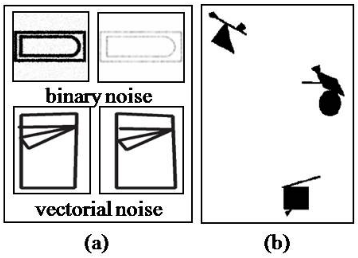

Fig. 1. Examples of synthetic document
(a) segmented symbol (b) random symbol set

The systems proposed by [9] and [2] support the generation of degraded images of segmented symbols as shown in the Figure 1 (a). 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 [9] [2] and vectorial [2]. The system described in [8] employs a complementary approach to build documents composed of multiple unconnected symbols. The Figure 1 (b) gives an example of document generated by this system. Each symbol is composed of a set of primitive (circles, lines, squares, etc.) randomly selected and mildly overlapped. They are next positioned on the image at a random location and without overlapping with the bounding boxes of the other symbols. Finally, noise is added to the generated images using a binary distortion method.

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 (engineering and architectural drawings, electrical diagrams, etc.) are composed of multiple objects constrained by spatial relations (connectivity, adjacency, neighbourhood, etc.). The design of a suitable process to build such documents is a challenging task. Indeed, realistic documents cannot 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 observes that the graphical documents are composed of two layers : a linear layer (the background) and a symbolic one. We use this property to build several document instances: *ie* symbol sets positioned in different ways using a same background as shown in the Figure 2. In this way, the building process of whole document is made easier and can be considered as a problem of symbol positioning on a document background.

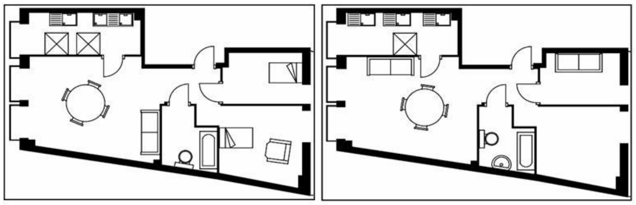

Fig. 2. Two document instances

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

2 Symbol Factory

Following the systems proposed by [9] and [2] we use geometrical primitives (straight lines, arcs and circles) and their associated thickness attributes to describe the symbol models. Each model is 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 wants to use. Obviously, in order to produce different document instances, these models are selected at random. The user controls the selection probabilities in the file of positioning constraints. Once selected we load the symbols from their model files,

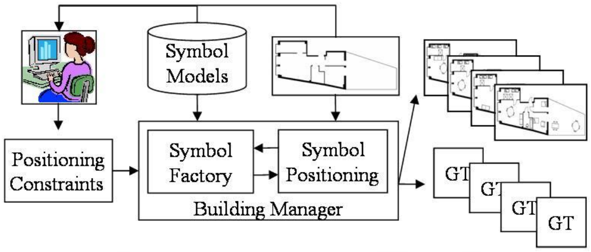

Fig. 3. Our system

scale them to adapt them to the background 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 presented in the next section.

3 Symbol Positioning

The goal of our system is to place randomly symbols on a given background. In order to do that, we use positioning constraints that will determine where and how the symbols could be placed. A natural way to define these constraints is to use some of the graphics primitives composing a symbol: these primitives can be exploited next to position the symbol on the background. Some examples could be the two connection points of a resistor, the top line of a bed, the two borders of a frame, etc. This definition makes complex the addition of new models in the system: the user has to define the constraints proper to a model before using. Also in order to position a symbol on the background, primitives corresponding to the constraints must be edited on the background. This process could take lot of time to the user in regard to the number of model and associated constraint. In our work we have considered another approach. We have defined generic constraints fully independent of models. The parameters of the constraints are computed in an automatic way function of models. It is then not necessary to worry about the models to handle during the edition of constraints.

Our constraints are taken at random from the symbols produced by the factory. The links between the constraints and the symbols are defined in the file of positioning constraints edited by the user. Next, the key mechanism of positioning the symbols on the background according to the constraints is detailed in the Figure 4. It raises on the matching between two points: a control point on the symbol and a positioning one defined on the background. The symbol is then positioned in order to fit the control point with the positioning one. To make more flexible our approach we have defined three possibilities to select the positioning points: using a fixed position on the background, or taking a random point in a geometrical shape. In the the first case the constraint defines a fixed value $(x; y)$ where the symbol must be positioned. In the second case, the constraint defines a geometrical shape were points could be selected at random and used for the positioning. We have used two types of shape, either a straight line (the point is selected at random along the line) or either a polygon (the point is selected at random inside the polygon). We will talk next about fixed, sliding and zone constraint to refer these three different positioning processes.

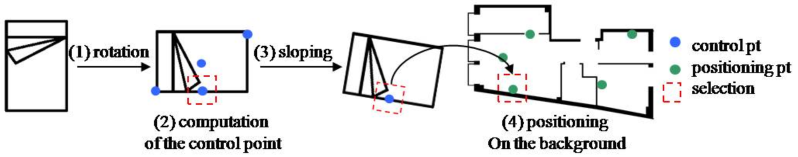

Fig. 4. Constraint mechanism

Also, in order to extend the positioning possibilities we also permit on a symbol to define a particular control point and to apply a rotation transformation. The symbol is then positioned in four steps: it is rotated (using a parameter that can be null, a fixed value or a range), its control point is computed, a slope parameter (between 0 and $2 \times \pi$) is used to incline both the symbol and the control point, and finally it is fixed on the positioning point using the control one. The key step of this process is the computation of the control point. This point is defined for every each constraint using 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 the bounding box to obtain a control point as explained in the Figure 5 (a). The α value is equal to $\theta \times 2\pi$, l is computed in different ways (1,2,3 and 4) according to the size of the bounding box sides and weighted at last by ρ. The Figure 5 (b) gives some examples of positioning around a 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 5 (c) and (d) gives examples using previous rotations of the symbol and with control points defined by $(\rho = 1.0, \theta = 0.25)$ and $(\rho = 1.0, \theta = 0.75)$.

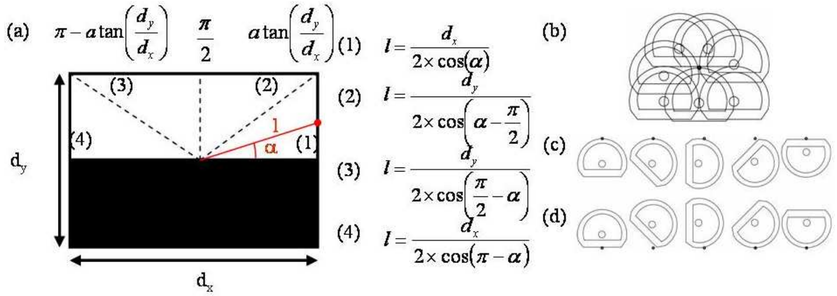
(a)
$\pi - a\tan\left(\frac{d_y}{d_x}\right)$
$\frac{\pi}{2}$
$a\tan\left(\frac{d_y}{d_x}\right)$
(1) $l = \frac{d_x}{2 \times \cos(\alpha)}$
(2) $l = \frac{d_y}{2 \times \cos\left(\alpha - \frac{\pi}{2}\right)}$
(3) $l = \frac{d_y}{2 \times \cos\left(\frac{\pi}{2} - \alpha\right)}$
(4) $l = \frac{d_x}{2 \times \cos(\pi - \alpha)}$
(3)
(2)
(4)
l
α
(1)
d_y
d_x
(b)
(c)
(d)

Fig. 5. Control point
(a) computation (b) (c) (d) examples of result

4 Building Manager

In the proposed system the factory and the positioning processes are managed by an explicit document building process. It starts with empty documents and fills them with symbols in a pseudo-random way. However, a positioning might fail. These failures appear for example when a symbol is positioned to overlap an existing one, when parts of a symbol overflow a constraint area, etc. The system must be able to identify these failures in order to cancel the positioning. Moreover, users might define constraints that could be hard to satisfy. The system must then detect these cases in order to avoid an infinite building process. To solve these problems our building manager uses five tests, four to check the positioning of symbols and one to stop the building process.

Our first positioning checking concerns the management of the free space of document. Indeed, during the building process several symbols can share the 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 6: 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 the new symbol we want to position with all the symbols already positioned on the document. Any positive case produces a building failure.

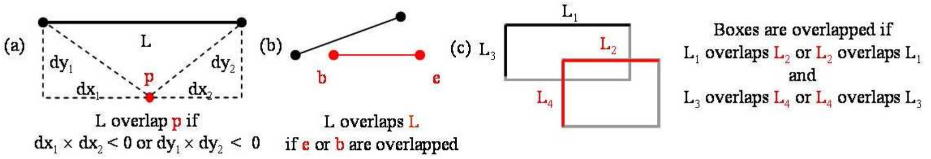
(a)
L
dy_1
dy_2
p
dx_1
dx_2
L overlap p if
$dx_1 \times dx_2 < 0$ or $dy_1 \times dy_2 < 0$
(b)
b
e
L overlaps L
if e or b are overlapped
(c)
L_1
L_2
L_3
L_4
Boxes are overlapped if
L_1 overlaps L_2 or L_2 overlaps L_1
and
L_3 overlaps L_4 or L_4 overlaps L_3

Fig. 6. Overlapping test
(a) line-point (b) line-line (c) box-box

Our second positioning checking deals with the sliding constraint as shown in the Figure 7. The positioning process of this constraint could produce overflows of symbols around the line borders (a). In order to limit the positioning to the line areas we have defined an overflow test. This test is based on the covering between two lines (b). It is just a logical adaptation of the overlapping test presented in the Figure 6. A symbol can be considered as overflowing if any of the borders $\{right, up, left, bottom\}$ of its bounding box is not covered by the constraint line L (c). A positive case produces then a building failure.

Our next 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 8 (a) gives an example of this case. It corresponds to a random fixed point generated too near of the borders of a polygon. In order to detect such a case we exploit the bounding box's corners of the symbol as explained in the Figure 8 (b). We test then if these corners are included in the polygon. Any false case will produce a building failure. This inclusion test is based on the method presented in the Figure 8 (c). This method sums

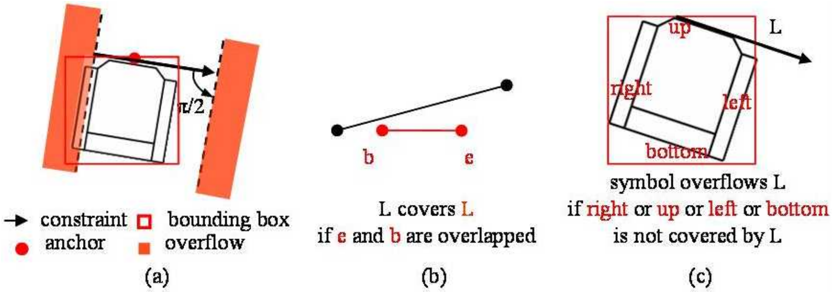

Fig. 7. Sliding checking
(a) symbol overflow (b) covering test (c) overflow test

the trigonometric angles of successive vectors joining the random point and the polygon ones. A $2 \times \pi$ value corresponds to an inclusion case.

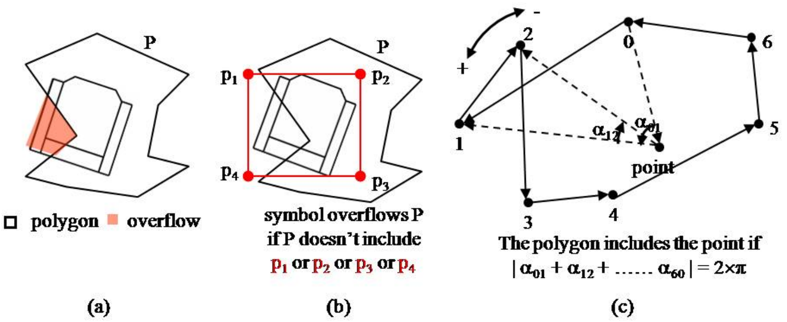
P
polygon
overflow
P
p_1
p_2
p_3
p_4
symbol overflows P
if P doesn't include
p_1 or p_2 or p_3 or p_4
-
+
2
0
6
1
α_{12}
α_{01}
point
5
3
4
The polygon includes the point if
$|\alpha_{01} + \alpha_{12} + \ldots\ldots \alpha_{60}| = 2\times\pi$
(a)
(b)
(c)

Fig. 8. Zone checking
(a) symbol overflow (b) overflow test (c) trigonometric inclusion test

We also check the number of positioned symbol per constraint. Indeed, for every each constraint a maximum number of symbol to position is defined. This number is one for a fixed constraint and can be larger for a sliding and a zone constraint. In this last case, it is defined by the user in the file of positioning constraints. During the building process, the system computes for each constraint the number of symbol already positioned. When this number becomes greater than the maximum a building failure is produced.

In the last test we control the progress of the building process in order to stop it if necessary. Indeed, the system must 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. This number corresponds to the sum of the maximum numbers of allowed symbol per constraint. If the number of building failures becomes greater than this number,we stop the process.

5 Experiments and Results

In this section we present some initial experiments and results of our system. The main objective of these experiments is to create databases of test document, with their corresponding ground-truth, for the series of the Symbol Recognition Contests[5]. To do it we have used the symbol model library defined for the previous editions of the Contests[5]. It 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 types. Obviously, the documents produced by our system are in a vector graphics form. For the Contest these documents should therefore be converted into binary images; noise can then be added by the distortion methods used in the past editions of the Contests [2].

We have edited a first set of constraint in order to build "bag of symbol" documents. The Figure 9 presents examples of these bags. In them the symbols are positioned 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 [8] (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 create an intermediate level of evaluation between the documents composed of a single segmented symbol (as proposed in the past editions of the Contest[5]) and whole documents (drawings, maps, diagrams, etc.).

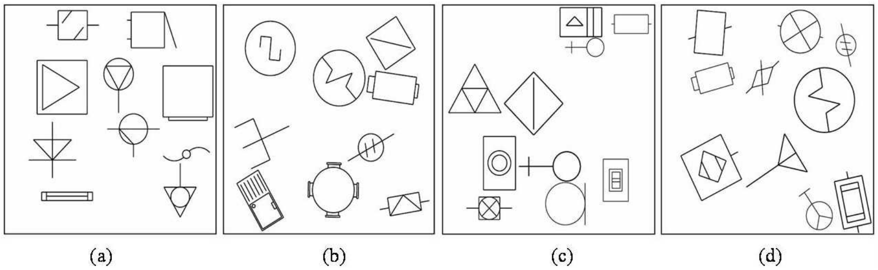

Fig. 9. Examples of bag of symbol
(a) none transformation (b) rotated (c) scaled (d) rotated & scaled

To generate these bags we have defined in our setting a single squared 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 editions[5] from 512×512 to 256×256 pixels. Based on this initial size we have generated bags of 1024×1024 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 9.

Using these size parameters we have generated 16 databases of 100 bags each. This corresponds to an overall number of 1600 bags composed of around 16000 symbols. These 16 databases have been generated by respecting the protocol used during the previous editions of the Contest[5]. 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 the Figures 9 (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 our experiments to the building of architectural drawings. The next Figure 10 presents some 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, etc.). However, the future edition of the Contest[8] will be a kickoff concerning the evaluation of whole documents. It will constitute an important gap for the systems and to limit it to a single domain seems to be fair. We have chosen the architectural drawings in recognition to their interesting properties concerning the connectivity and the orientation of symbols.

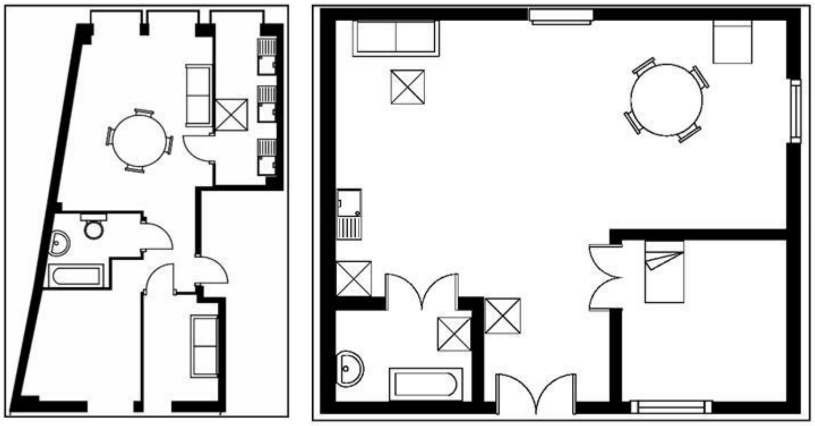

Fig. 10. Examples of built architectural drawing

To generate these drawings we have retained the size parameter defined for the bags: 256×256 pixels per symbol. Obviously, the use of filled backgrounds makes the images bigger in regard to the one of bags. In order to produce drawings of reasonable dimensions we have fixed a limit of about 4096^2 pixels per image by considering only the backgrounds composed of a small number of rooms (from 4 to 8). We have then selected 10 real-life drawings and created the backgrounds by cleaning their text and symbol parts with an image editor. Using these backgrounds we have defined sets of constraint in order to generate databases of 100 images per background and with 14 to 28 symbols per image. This corresponds to an overall number of 1000 drawings composed of around 18 000 symbols. Obviously, to generate these drawings we have selected only the architectural models of the Contest library[5]. It corresponds to an overall

[8] In 2009 at La Rochelle city (France)

number of 16 models. The Figure 11 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 256×256 pixels.

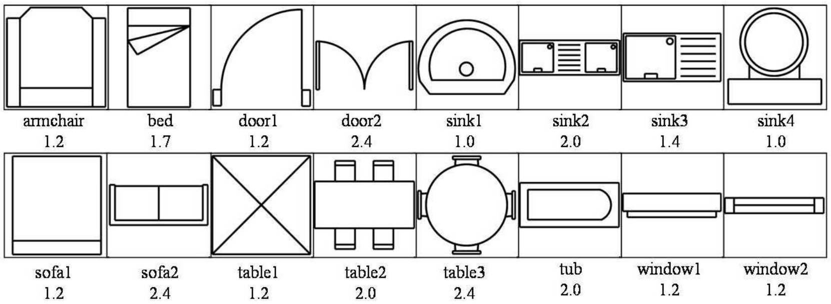

Fig. 11. Architectural symbols(labels & resizing parameters)

We have then used these models and the resizing parameters in the constraints. The number of constraints per background is about 20. These constraints can be of fixed, sliding or zone type. We have used the fixed constraint to position the door and the 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 positioned in the direction of the line 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 furniture elements 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 for the building of 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, etc.). To do it we have exploited the layer property of graphical documents in order to position symbol sets in different ways using 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 show how our system allows to produce large databases of document that look real.

Concerning future perspectives different works are planned. In the short term we plan to develop a GUI to edit the positioning constraints. It will speed up the editing process and help users to build their own databases. Also, based on this GUI we want to use our system to generate other kinds of document like electrical drawings or geographical maps. A more long-term perspective concerns the development of a performance characterization method. Such methods are now required in order to compare the system results with the ground-truth. However, when we work with whole documents the characterization becomes harder because it has to be done between symbol sets. These symbol sets can be of different size, and large gaps can also appear concerning the locations of symbols. Different matching cases can appear and the characterization method should be able to detect and handle them properly.

7 Acknowledgements

The authors wish to thank Karim Zouba and Murielle Ramangaseheno (LITIS, Rouen University, France) for their contributions to this work. This work was funded by the Spanish Ministry of Education and Science under grant TIN2006-15694-C02-02, and supported by the EPEIRES[5] project of the French Techno-Vision program 2005.

References

1. Kasturi, R., Phillips, I.: The first international graphics recognition contest-dashed-line recognition competition. In: Workshop on Graphics Recognition (GREC). Volume 1072 of Lecture Notes in Computer Science (LNCS). (1996)
2. Valveny, E., al.: A general framework for the evaluation of symbol recognition methods. International Journal on Document Analysis and Recognition (IJDAR) **1**(9) (2007) 59–74
3. Lopresti, D., Nagy, G.: Issues in ground-truthing graphic documents. In: Workshop on Graphics Recognition (GREC). Volume 2390 of Lecture Notes in Computer Science (LNCS). (2002) 46–66
4. Wenyin, L., Dori, D.: Principles of constructing a performance evaluation protocol for graphics recognition algorithms. In: Performance Characterization and Evaluation of Computer Vision Algorithms. Springer Verlag Publisher (1999) 97–106
5. Chhabra, A., Phillips, I.: The second international graphics recognition contest - raster to vector conversion : A report. In: Workshop on Graphics Recognition (GREC). Volume 1389 of Lecture Notes in Computer Science (LNCS). (1998) 390–410
6. Phillips, I., Ha, J., Haralick, R., Dori., D.: The implementation methodology for the cd-rom english document database. In: International Conference on Document Analysis and Recognition (ICDAR). (1993) 484–487
7. Wenyin, L., Zhai, J., Dori, D.: Extended summary of the arc segmentation contest. In: Workshop on Graphics Recognition (GREC). Volume 2390 of Lecture Notes in Computer Science (LNCS). (2002)
8. Aksoy, S., al.: Algorithm performance contest. In: International Conference on Pattern Recognition (ICPR). Volume 4. (2000) 870–876
9. Zhai, J., Wenyin, L., Dori, D., Li, Q.: A line drawings degradation model for performance characterization. In: International Conference on Document Analysis And Recognition (ICDAR). (2003) 1020–1024